EINE BEILAGE
IN CHIP NR. 2
FEBRUAR 1990

CHIP PLUS

OBJEKTORIENTIERTE PROGRAMMIERUNG

Mehr als ein Modetrend III
Die Pascal-Weiterentwicklung IV
Vergleich Quick-Pascal gegen Turbo Pascal IX
Überleben herkömmliche Programmiersprachen? XIV
OOP und RDBMS verknüpfen VIII
Datenbank mit Objektverwaltung und Regelsystem XVII
Das leistet C++ XX

CHIP SPECIAL

TURBO-PASCAL

DIE NUMMER EINS IN TURBO-PASCAL

Die kompetente Reihe, in allen Pascal-Fragen.

Praktisch von Anfang an begleiteten die Sonderhefte CHIP-SPECIAL den Siegeszug von Turbo-Pascal. Jede Ausgabe enthält wertvolle Informationen, Tips und Tricks, Musterlösungen – bis hin zu kompletten Programmprojekten. „Learning by doing" ist die Devise. Die abgedruckten Programme und Module können zwar selbstverständlich professionell eingesetzt werden, der Leser soll die Arbeitsweise aber auch verstehen. Ein besonderer Schwerpunkt liegt daher in den Einsteiger- und Fortgeschrittenenkursen aus den verschiedensten Gebieten.

NEU

TURBO-PASCAL, Ausgabe 13

Auch dieses Heft wird auf Grund der Angebotsvielfalt ein Renner wie auch die anderen Turbo-Pascal-Hefte.
Best.-Nr.: 0994, DM/sfr 28,–, öS 230,–

TURBO-PASCAL, Ausgabe 12

Der Hammer dieser Ausgabe: Eine TSR-Unit, mit der Sie speicherresidente Programme à la Side-Kick in Turbo-Pascal schreiben können. Zudem: Simulation mit Parallel-Programmierung, Long-Jumps – Sprünge über Blockgrenzen und neue Features von Turbo-Pascal 5.0. Wie angekündigt: Ein BGI-Editor zum Selbstgenerieren von Vektorzeichensätzen. Ordnung in die private Videothek bringt Video.PAS.

Aus den Inhalten der bereits erschienenen Ausgaben:

Turbo Pascal, Ausgabe 1: PLIST – der Profilister, komplette Adreßverwaltung...
Turbo Pascal, Ausgabe 2: Der erste Makro-Assembler der Welt für 8088/86, 8080/85, Z80 und 6502...
Turbo Pascal, Ausgabe 3: Window-Technik, Neue Bibliotheks-Module...
Turbo Pascal, Ausgabe 4: Große SPECIAL-Toolbox...
Turbo Pascal, Ausgabe 5: TDB-2, die komplette Turbo-Datenbank...
Turbo Pascal, Ausgabe 6: Statistik-Grundlagen und Anwendungen, Mathematik...
Turbo Pascal, Ausgabe 7: Alles über Turbo-Pascal Version 4.0...
Turbo Pascal, Ausgabe 8: Coroutinen und nebenläufige Prozesse...
Turbo Pascal, Ausgabe 9: Programmierpraxis und viele neue Anwendungen...
Turbo Pascal, Ausgabe 10: DFÜ-Programm, Grundlagen der Parallel-Programmierung...
Turbo Pascal, Ausgabe 11: Was Turbo-Pascal 5.0 bringt, DFÜ-Utilities...

Die Turbo-Pascal-Ausgaben 1–12 kosten jeweils DM/sfr 28,– bzw. öS 230,–.

DAS ALLERBESTE AUS DER TURBO-REIHE

Die CHIP-SPECIAL Toolbox für Turbo-Pascal und Quick-Pascal komplett mit allen Sources auf Diskette.

Mit nur wenigen Anweisungen können Features wie Fenstertechnik, Menüauswahl, Maskeneingabe und sogar ein kompletter Editor in eigene Programme integriert werden. Der Einsatz komplexer Datenstrukturen wie Bäume und Listen wird durch die mächtigen Tools so einfach wie das Arbeiten mit Feldern. Außerdem wurden parallele Prozesse und dynamische Strings implementiert. Alle Units sind sehr ausführlich beschrieben, Kurse aus verschiedenen Themengebieten vermitteln das nötige Hintergrundwissen für den nutzbringenden Einsatz der Module.
Ein unbedingtes Muß für jeden Pascal-Programmierer – vom ambitionierten Einsteiger bis zum Profi.
Best.-Nr.: 1008, DM/sfr 49,–, öS 380,–

Alle Programme aus den CHIP SPECIAL Turbo-Pascal 1–13 sind auf Diskette erhältlich. Die Auslieferung der Software erfolgt über den Elsa-Data CHIP Shop. Bitte beachten Sie, daß die Dokumentation zu den Disketten in der entsprechenden CHIP SPECIAL Ausgabe zu finden ist.

Für Ihre Bestellung verwenden Sie bitte die Bestellkarten aus diesem Heft. Für ganz Eilige gibt es den telefonischen Tag- und Nacht-Bestellservice: 0931/4182283.

Weitere Special-Angebote finden Sie in diesem Katalog. Am besten noch heute kostenlos beim Verlag anfordern.

Mehr als ein Modetrend

Ulrich Parthier

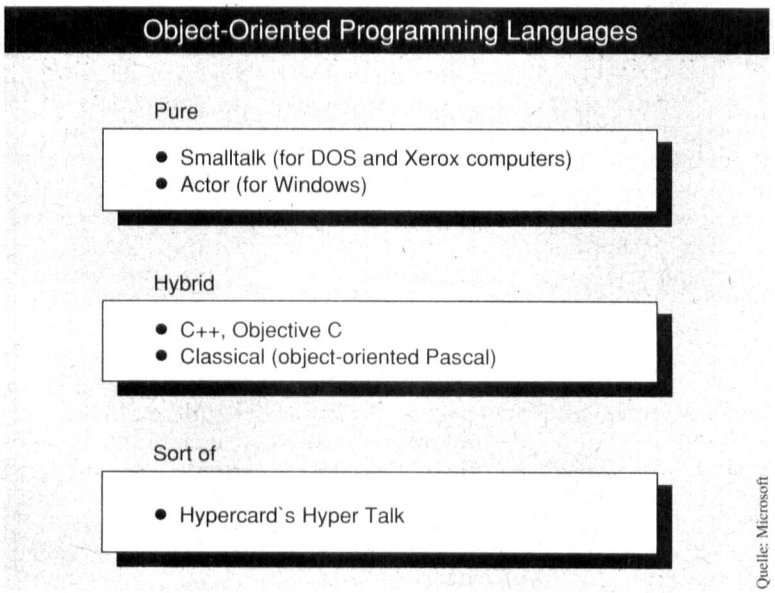
Wie läßt sich der Programmieraufwand auf ein Minimum reduzieren, ohne die Qualität zu vernachlässigen? Das ist die zentrale Frage, die sich Entwickler wie auch DV-Verantwortliche ständig stellen müssen.

Nun ist das Thema objektorientierte Programmierung (OOP) ja beileibe nicht neu. 1967 wurde OOP mit der Entwicklung der Programmiersprache Simula von Nygaard und Dahl erfunden. Kurze Zeit später folgte das heute noch gebräuchliche Smalltalk, dessen Ursprung im Xerox Palo Alto Research Center liegt. In diesem Forschungszentrum steht übrigens auch die Wiege der Apple-Benutzeroberfläche. Die Wurzeln reichen zurück bis in die sechziger Jahre, als man schon mit abstrakten Datentypen arbeitete.

Eine interessante Antwort auf die Frage, warum sich denn die objektorientierte Programmierung erst jetzt durchzusetzen beginnt, gab Tom DeMarco, der „Vater" der strukturierten Analyse, vor kurzem auf einem Seminar zum Thema CASE. Er merkte an, daß jeder neue Trend in der Software-Industrie bis zu 20 Jahre benötige, um sich durchzusetzen.

Allerdings, so DeMarco weiter, genüge in puncto objektorientierter Programmiersprachen keine der heutigen Sprachen den Anforderungen der Anwender. Beispielsweise seien weder Object C noch C++, noch das miserabel implementierte Smalltalk technische Highlights. Derzeitige Programmiersprachen mit objektorientierten Erweiterungen haben trotzdem eine gute Chance am Markt. Und so wie Mitte der siebziger Jahre der Siegeszug der strukturierten Programmierung begann, so gehören die neunziger Jahre nach Meinung der Experten ohne jeden Zweifel der objektorientierten Programmierung.

Neben den 1989 von Borland vorgestellten Turbo-Pascal, Version 5.5 und Microsofts Quick-Pascal, Version 1.0 gibt es weitere gute Neuigkeiten. Ein Problem der Datenbankentwickler bestand bisher in ihrer Eingeschränktheit hinsichtlich der Datentypen. Hier hat die Ingres Corp., bekannt durch ihr gleichnamiges verteiltes relationales Datenbank-Management-System, Akzente gesetzt – ähnliches ist von Informix Software zu berichten.



Klassifizierung der objektorientierten Sprachen in drei Klassen

Die Pascal-Weiterentwicklung

Alfred Ertl

Der Autor Alfred Ertl ist Produkt-Manager Turbo-Pascal bei Borland.

An die Vorstellung von Borlands Turbo-Pascal 5.5 sind viele Erwartungen geknüpft worden: Die Version 5.5 ist nicht nur voll Quellcode-kompatibel zu den Versionen 4.0 und 5.0 – sie führt in die Welt der objektorientierten Programmierung ein.

Objektorientiertes Programmieren heißt: Code und Daten werden zu kleinen Einheiten, sogenannten Objekten, zusammengefaßt, die systematische und praktische Vorteile haben. Sie weisen Eigenschaften auf, die sie auf andere Objekte vererben, und reagieren auf Programmbefehle entsprechend ihren Eigenschaften: Wenn ein Quadrat sich selbst zeichnen soll, tut es etwas anderes als ein Kreis.

Bisher war OOP vom Erlernen einer neuen Sprache abhängig – etwa C++ – und für Apples Objekt Pascal wäre sogar ein neuer Computer fällig gewesen. Turbo-Pascal 5.5 beendet jetzt dieses Dilemma.

Die objektorientierte Programmierung ist nicht ganz leicht zu erklären. Es handelt sich dabei um das Zusammenspiel verschiedener wichtiger Konzeptionen, die auf unterschiedlichen Ebenen aufeinander wirken. Wie bei allen Hochsprachenkonzeptionen geht es auch bei der objektorientierten Programmierung darum, die Komplexität der Software zu bewältigen. Dabei wird eine große vereinende Theorie der komplexen Anwendung übergeordnet, indem Programmcode und Daten zu einer Hierarchie von Strukturen – Objekte genannt – verbunden werden.

Objekte

Ein Objekt ist das Modell für die Komponente eines Software-Problems. Dieses Modell benutzt eine Metapher, die widerspiegelt, wie die Bestandteile der realen Welt vorkommen und wie sie aufeinander einwirken. Die Welt besteht aus Objekten, wie Motorrädern, Mikroskopen usw. Innerhalb von Objekten existieren Teilobjekte. So besteht ein Motorrad aus Rädern, Getriebe, Bolzen und vielen anderen Objekten, die wiederum aus vielen anderen Objekten bestehen und so weiter. Jeder Bestandteil eines Motorrads ist genauestens definiert und verfügt sowohl über verschiedene Eigenschaften als auch über unterschiedliche Methoden des Betriebs.

Ein Motorrad stellt eine Hierarchie von Teilen dar, die sich in weitere Bestandteile untergliedern lassen. Alle diese Teile arbeiten entsprechend einem Gesamtplan vorhersagbar zusammen. Ist ein Teil abgenutzt, so bereitet es aufgrund des standardisierten Charakters der Teile keine Mühe, dieses Teil zu entfernen und durch neue zu ersetzen.

Wir wollen, daß auch unsere Software auf diese Weise arbeitet. Sie sollte mindestens genausogut oder eher noch besser als ein Motorrad arbeiten, denn Prozeduren nutzen sich nicht ab wie Bremsbeläge.

In der bisherigen Praxis ist der Gesamtplan aber meist undurchsichtig, die einzelnen Software-Teile passen nicht immer gut zusammen, und nur einige von ihnen sind in irgendeiner Weise standardisiert. Objektorientiertes Programmieren kann dies alles verändern.

Vererbung und die Taxonomie der Software-Bestandteile

Die Wissenschaft hat eine eigene Systematik entwickelt, um die komplexen Erscheinungen der Welt zu beschreiben: die Taxonomie. Lebende und nicht lebende Dinge werden danach klassifiziert, worin sie sich gleichen und worin sie sich unterscheiden. Der vielleicht aufregendste Aspekt der objektorientierten Programmierung ist die Vererbung, die eigentlich nichts anderes ist als die Taxonomie von Software-Bestandteilen.

Entomologen klassifizieren Insekten nach einer großen Tabelle von Insektenfamilien und -klassen. Unterhalb der Hauptkategorie „Insekt" existieren Untergruppen, zum Beispiel Insekten mit Flügeln, Insekten ohne Flügel. Unterhalb von Insekten mit Flügeln gibt es weitere, noch spezifischere Untergruppen: Fliegen, die über ein Paar Flügel verfügen, und andere Insekten mit zwei Paar Flügeln. So geht es immer weiter bis zu den Hunderttausenden von individuellen Insektenspezies.

Insekten einer Gruppe verfügen alle über die Eigenschaften, die für diese Gruppe festgelegt sind. Mücken und Moskitos sind beide Fliegen; sie erben ihr „Fliege-Sein" von der höheren Kategorie „Fliegen", da sie beide ein paar Flügel besitzen. Sie werden zwar durch differierende Verhaltensweisen unterschieden (Moskitos stechen, Mücken nicht), können aber beide gleichermaßen als „Fliegen" bezeichnet werden. An anderer Stelle in der Hierarchie liegen die Schmetterlinge, die keine Fliegen sind, weil sie zwei Paar Flügel besitzen. Sie erben die Eigenschaft des „Geflügeltseins" von der höheren Kategorie der geflügelten

Insekten und sind darin den Fliegen ähnlich. In jeder anderen Hinsicht aber sind sie von den Fliegen verschieden.

„Vererbung" bei Software-Objekten

Software-Objekte bilden eine ähnliche Hierarchie. Ein Rechteck auf dem Bildschirm kann als Fenster bezeichnet werden. Es hat eine X,Y-Position, eine Breite und eine Höhe und vielleicht eine bestimmte Einfassungsart und Hintergrundfarbe, sonst jedoch nichts. Neben Fenster könnte in der Tabelle auch Menü stehen, welches über alles verfügt, was Fenster hat, zudem aber noch verschiedene Zeilenpunkte und einen Rollbalken besitzt. Des weiteren könnte neben Fenster noch ein Editor existieren, der alle Merkmale von Fenster teilt, zusätzlich aber auch Zeichen aus der Tastatur empfangen und bearbeiten kann.

Sowohl Menü als auch Editor erben die Eigenschaften von Fenster: eine Position auf dem Bildschirm, eine Breite und eine Höhe sowie eine bestimmte Einfassungsart und Hintergrundfarbe. Beide können gleichermaßen Fenster genannt werden, beide unterscheiden sich jedoch voneinander darin, wie sie aussehen und was sie tun.

In unserer Objekttabelle könnte unterhalb von Editor etwas mit dem Namen Feld plaziert werden, das alles hat, was einen Editor ausmacht, gleichzeitig aber auf eine einzige Textzeile beschränkt ist und am Anfang einen Prompt-String hat.

Im Jargon der objektorientierten Programmierung sind Fenster, Editor und Feld Objekttypen. (Einige objektorientierte Programmsysteme, im besonderen Smalltalk, bezeichnen sie als Klassen. „Objekttypen" und „Klassen" sind identisch.) Objekttypen existieren innerhalb einer Hierarchie, in der Vererbungsbeziehungen zwischen den einzelnen Objekttypen definiert sind. Die Typen selbst sind keine Objekte, sondern eher Schablonen zur Entwicklung von Objekten. Bei Bedarf wird ein Beispiel eines Objekttyps (gewöhnlich als Objekt bezeichnet) entwickelt und angewendet.

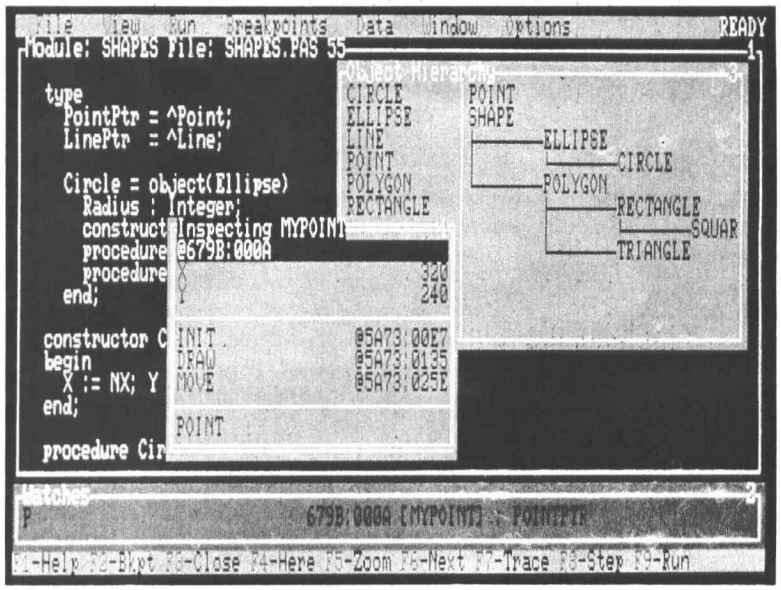
Vorzüge der Vererbung

Um zu verstehen, warum die Vererbung so wichtig ist, muß man wissen, daß alles an einem Objekt vererbt wird. Dies schließt Daten wie X,Y-Koordinaten und auch Code ein, der das Objekt „zum Arbeiten" bringt. Der Code innerhalb eines Objekts ist eng an dieses Objekt in Form von Prozeduren, den Methoden gebunden.

Um beim oben genannten Beispiel zu bleiben: Der Objekttyp Fenster verfügt vielleicht über eine Methode, die das Fenster über den Bildschirm bewegen kann. Menü erbt diese Methode vom Fenster. Da Menü alles ist, was auch ein Fenster ist, kann der Kode, der ein Fenster bewegen kann, genausogut ein Menü bewegen.

Wiederverwendbarkeit des Codes

Menü braucht also, um sich selbst zu bewegen, keinen Code neu zu implementieren. Es kann zu diesem Zweck den gleichen Code, den es von Fenster erbt, verwenden. Diese sogenannte Wiederverwendbarkeit des Codes leistet mehr als nur die Vermeidung unnötiger Code-Duplizierung. Sie stellt sicher, daß eine korrekt implementierte Funktion auch korrekt durch die Objekttypen-Hierarchie weitergegeben wird. Ist eine Methode, die Fenster korrekt über den Bildschirm bewegt, einmal geschrieben, werden alle Objekttypen, die von Fenster abstammen, sie genauso korrekt über den Bildschirm bewegen.



Objekthierarchien im Turbo-Debugger. So behält man auch bei komplizierten Hierarchien den Überblick.

Einheitliches Erscheinungsbild

Außerdem werden sich sämtliche Objekttypen gleich über den Bildschirm bewegen: Die Anwendung als Ganzes erhält ein einheitlicheres Erscheinungsbild und zeigt einen klaren Zusammenhang zwischen ihren Bestandteilen. Jeder funktionelle Teil des Programms wird einmal an einem passenden Ort innerhalb der Objekthierarchie implementiert und pflanzt sich dann durch die Hierarchie fort bis zu allen Objekten, die ihn benötigen. Dies bedeutet, daß sämtliche Menüs eine einheitliche Form haben; Bildrollbalken arbeiten überall, wo sie erscheinen, identisch. Die Anweisungen zur Feld-Ausgabe sind für alle Felder gleich. Die gesamte Anwendung erhält ein sauberes, einheitliches Erscheinungsbild.

Frühe und späte Bindung

Die Vererbung ist die einfachere der beiden entscheidenden Neuerungen, die die objektorientierte Programmierung bie-

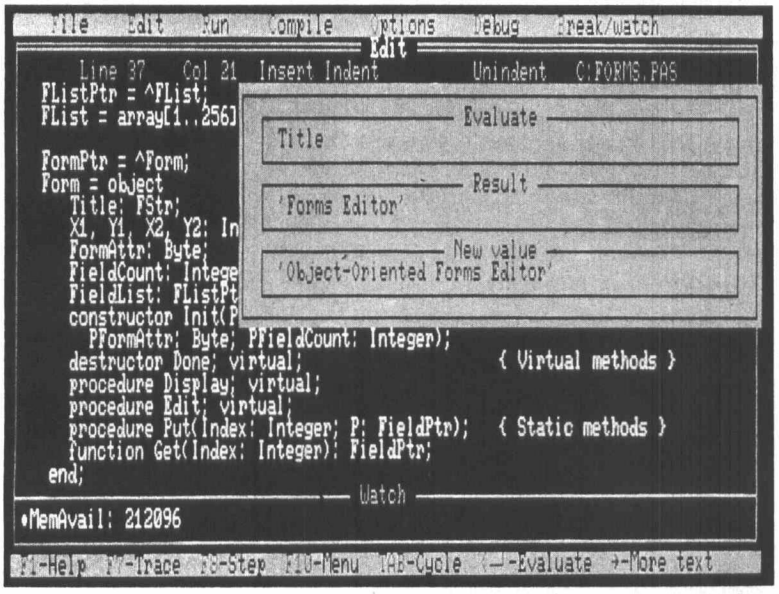
tet. Die zweite Neuerung ist die sogenannte späte Bindung. Sie bringt noch mehr Leistung, ist aber auch schwieriger zu erklären.

Bindung ist der Punkt im Leben eines Programms, an dem der Aufrufer einer Prozedur die Adresse dieser Prozedur erhält. Bei früher Bindung geschieht dies, wenn das Programm kompiliert und gelinkt wird. Tatsächlich verwenden alle traditionellen Programmiersprachen, einschließlich C und Pascal, ausschließlich die frühe Bindung. Die frühe Bindung setzt voraus, daß die Typen aller Variablen zur Zeit der Kompilation bekannt sind und sich danach nicht verändern. Dies ist ein Charakteristikum der Sprachen mit starker Typenbetonung.

Bei der späten Bindung ist im Gegensatz dazu der Aufrufer erst dann an die Prozedur gebunden, wenn der Prozeduraufruf tatsächlich erfolgt. Es gibt verschiedene Wege, dies zu implementieren. Wie wirkungsvoll der Prozeß wird, hängt ausschließlich von der Geschicklichkeit des Implementierenden ab.

Objekte im Griff: Auch der integrierte Debugger in Turbo Pascal 55. kennt Objekte.

Die späte Bindung gibt dem Programmierer ein mächtiges Werkzeug in die Hand. Bestes Beispiel dafür ist die Verarbeitung von Objektsammlungen, deren genaue Typen zum Zeitpunkt der Kompilation nicht bekannt sind. Das Problem könnte man übertragen so ausdrücken: „Hier ist ein Marmeladenglas, in dem sich sieben Insekten verschiedener Spezies befinden. Sie alle sollen als Ausstellungsstücke präpariert werden." Sicherlich können alle Insekten irgendwie präpariert werden, aber die Art und Weise ist bei jedem anders: Bei geflügelten Insekten müssen die Flügel ausgestreckt, Wasserinsekten müssen chemisch getrocknet, mikroskopisch kleine Insekten müssen auf einem Objektträger in Balsam eingebettet werden etc. Wird das Problem benannt (das heißt zum Zeitpunkt der Kompilation), so wissen wir nur, daß wir einige Insekten präparieren müssen. Bei der Lösung des Problems (das heißt zur Laufzeit) müssen wir die exakte Spezies jedes Insekts identifizieren und jedes seinen Bedürfnissen entsprechend präparieren. Die zu wählende Prozedur für die Präparation wird nicht eher festgelegt, bis wir de facto darangehen, jedes einzelne Insekt zu präparieren. Genauso kann man sich die späte Bindung vorstellen.



Späte Bindung wird dadurch möglich, daß die objektorientierte Programmierung den Code mit den Daten verschweißt. Jedes Objekt „weiß", wie man bestimmte Dinge tut, weil es über eine Reihe von speziell dafür entwickelten Methoden verfügt. Wenn ein Objekt Bedürfnisse hat, die für es selbst spezifisch sind, so besitzt es eine Methode, um diese Bedürfnisse zu befriedigen. Eine der Eigenarten objektorientierten Programmierens ist, von Objekten so zu reden, als wären sie kleine Roboter, die auf Befehle – Nachrichten genannt – reagieren können. Statt: „Laß uns diese Prozedur mit diesen Daten ausführen" würde die objektorientierte Programmierung sagen: „Objekt, führe jetzt deine Methode – zum Beispiel Präparieren – aus." Ein Objekt und seine gesetzmäßigen Abläufe sind also in einer Einheit zusammengeschweißt. Die technische Bezeichnung für dieses Zusammenschweißen von Code und Daten heißt Kapselung.

Späte Bindung und Polymorphie

Ein exzellentes Beispiel für die Notwendigkeit einer späten Bindung ist unser angeführtes Beispiel einer Objekthierarchie, die die Typen Fenster, Menü, Editor und Feld umfaßt. In jedem System mit sich überschneidenden Fenstern wird ein Fenster oft Teile eines anderen Fensters überlappen. Wird das obere Fenster entfernt, so muß das darunterliegende Fenster neu gezeichnet werden, um die Teile, die überschrieben waren, wiederherzustellen.

Es ist klar, daß ein Menü nicht auf die gleiche Weise wie ein Feld gezeichnet wird. Jedes Objekt in der Hierarchie muß also wissen, wie es speziell sich selbst wieder neu zeichnen kann. Jedes Objekt wird seine eigene gesonderte Methode benötigen, die seine Bildschirmdarstellung wieder aufbaut. Muß ein Objekt also neu gezeichnet werden, so braucht die Anwendung nicht einmal zu wissen, welcher Typ von Fenster überschrieben wurde und nun neu gezeichnet werden muß. Die Anwendung sendet dem überschriebenen Fenster einfach die Nachricht „Neuzeichnen". Das Fenster führt dann seine ganz spezielle Methode dafür aus und zeichnet sich selbst in korrekter Weise neu auf den Bildschirm.

Sind in einer Anwendung sieben verschiedene Arten von Fenstern vorhanden, so wird es sieben verschiedene Methoden geben, um diese Fenster neu zu zeichnen, jede mit dem gleichen Namen „Neuzeichnen." Die auszuführende Aufgabe ist immer die gleiche – das Neuzeichnen eines Fensters. Sie erfolgt aber bei jedem Objekt in einer speziell auf seine Bedürfnisse zugeschnittenen Art und Weise. Diese Vielfältigkeit der Formen für einen einzelnen Methodennamen wird als Polymorphie (griech.: viele Formen, gestalten) bezeichnet. Die Polymorphie ist ein extrem leistungsstarkes Instrument zur Verallgemeinerung einer einzelnen Aufgabe für viele verschiedene Objekte. Durch die späte Bindung wird die Polymorphie möglich.

Erweiterung von objektorientiertem Code

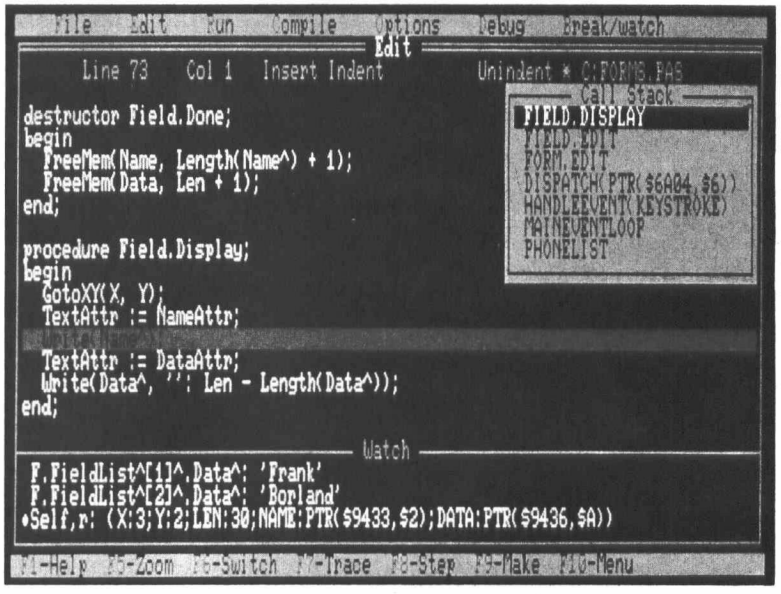
Die Kombination von Vererbung und später Bindung hat einen entscheidenden Vorteil gegenüber herkömmlichen Programmiermethoden: Der Anwender von objektorientiertem Code kann diesen erweitern, ohne im Besitz des Quellkodes zu sein.

Die Vererbung arbeitet über die Grenzen der Kompilationszeit hinaus – das heißt, solange ein Programmierer eine Beschreibung der Schnittstelle zu einem Objekttyp besitzt, kann er oder sie einen Nachkommen-Objekttyp festlegen, der all das erbt, was dessen Vorfahren-Objekttyp auszeichnet. Dies gilt auch dann, wenn der Vorfahre kompiliert wurde, bevor der Nachkomme geschrieben oder überhaupt ausgedacht wurde. Da die späte Bindung während der Laufzeit erfolgt, können Methoden, die noch nicht existierten, als ein Codemodul kompiliert wurde, aus diesem Modul heraus aufgerufen werden, indem der oben beschriebene Polymorphiemechanismus zum Tragen kommt.

Die Erweiterbarkeit erlaubt den Verkauf oder gemeinsamen Gebrauch von Objektbibliotheken, ohne daß damit Algorithmen oder Geschäftsgeheimnisse preisgegeben werden müßten. Außerdem ist eine Anwendung, die eine Objektbibliothek verwendet, nicht unbedingt durch die vorhandenen Spezifikationen der Objekte in der Bibliothek begrenzt. Wenn eines oder mehrere Bibliotheksobjekte nicht ganz den Bedürfnissen des Programmierers entsprechen, so können diese Objekte nachträglich modifiziert werden, indem sie zu neuen Objekttypen erweitert werden.

Objektorientierte Programmierung unter DOS

Die objektorientierte Programmierung hat die Hauptströmung der DOS-Programmierung noch nicht erreicht. Gegenwärtig wird meistens in speziellen Sprachen objektorientiert programmiert, beispielsweise in Smalltalk, Simula und Eiffel, von denen einige wenige ohne weiteres für DOS-Programmierer verfügbar sind.



Turbo-Pascal 55. beim Debuggen einer Methode. Die Aufrufreihenfolge zeigt das Fenster „call stack" an.

Aber dies ändert sich bereits. C++, die objektorientierte C-Erweiterung von Bjarne Stroustrup, hat sich im Bereich der Workstations und Minicomputer durchgesetzt und beginnt nun auch in DOS-Implementationen aufzutauchen. C++ unterscheidet sich von den eher forschungsmäßig ausgerichteten objektorientierten Programmiersprachen (etwa Smalltalk) dadurch, daß es von Anfang an mehr dafür entwickelt wurde, als Compiler implementiert zu werden statt als Interpreter. Der von C++ erzeugte Code ist leistungsmäßig dem von traditionellen C-Compilern produzierten Code fast gleichzusetzen, weil diese Sprache speziell für eine effiziente Code-Erzeugung entwickelt wurde.

Apple Computer definierte 1985 eine Pascal-Erweiterung mit Objektmerkmalen. Objekt-Pascal ist in der Macintosh-Welt weit verbreitet und wird langsam auch für DOS realisiert. Ähnlich C++ wurde Objekt-Pascal als Compiler implementiert und verringert nicht die Leistung des erzeugten Codes durch erhöhten Aufwand.

Sowohl C++ als auch Objekt-Pascal heben sich von den Spezialsprachen durch eine wichtige Eigenschaft ab: Rückwärtskompatibilität zum industriellen Programmcode, der in C und Pascal geschrieben wurde. Das Hinzufügen von objektorientierten Techniken zu einem existierenden Programm ist wesentlich einfacher als das Neuschreiben der Anwendung in einer völlig neuen Sprache. Finanzielle Mittel für die Aus- und Fortbildung können so eingespart werden. Der Weg zum Objektfachwissen ist kürzer und weniger beschwerlich. ■

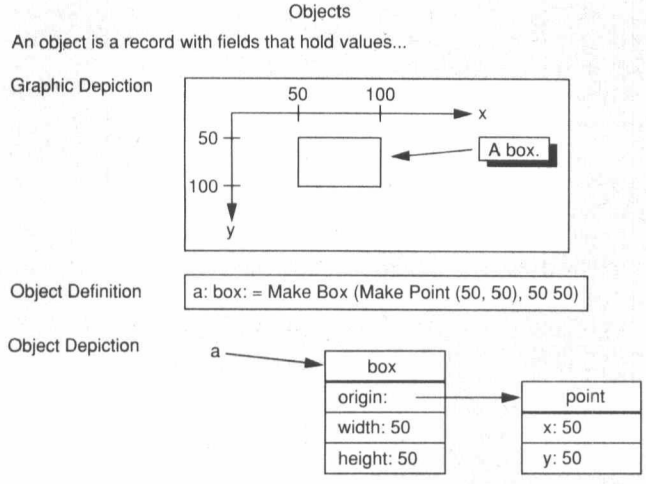
OOP und RDBMS verknüpfen

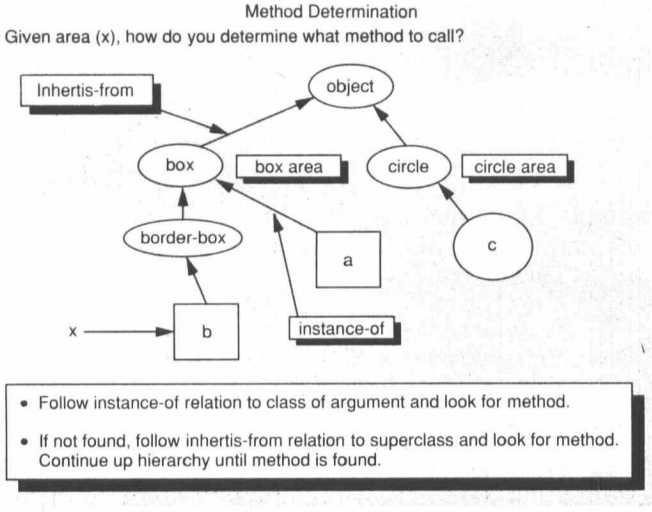
Prof. Larry Rowe

Der Autor Professor Larry Rowe forscht und lehrt an der Universität Berkeley. Außerdem ist er Vice President Advanced Development der Ingres Corp. in Alameda/ Kalifornien.

Anwendungs-Entwicklungsumgebungen der nächsten Generation werden auf objektorientierten Konzepten basieren. Hierbei werden künftig verstärkt Datenbank-Management-Systeme für die Speicherung und Verwaltung von Objekten (Daten und Methoden) zum Einsatz kommen.

Ziel des objektorientierten Ansatzes, egal ob es um die Datenhaltung oder die Programmierung geht, ist die Erhöhung der Produktivität in der Software-Entwicklung. Ein Hauptvorteil der objektorientierten Programmierung (OOP) besteht durch die dadurch mögliche Modularisierung von Software. Zudem ist Rapid Prototyping, also das Erzeugen von Vorläuferanwendungen, durch Objektorientierung wesentlich einfacher.





Es ist zu erwarten, daß herkömmliche Programmiersprachen wie C, Pascal, Lisp und andere künftig verstärkt um objektorientierte Elemente erweitert werden. Die OOP-Sprachen C++ und Common Lisp Object System (CLOS) haben bereits begonnen, sich zu etablieren.

Allerdings richten sich traditionelle Programmiersprachen, die um OOP-Merkmale bereichert wurden, in erster Linie an den einzelnen Entwickler. Für Software-Entwicklungsteams sind diese Single-User-Umgebungen nicht geeignet, da der gemeinsame Zugriff auf Objekte und Objektklassen durch mehrere Benutzer nicht geregelt ist. In Zukunft werden relationale Datenbanksysteme die ideale Grundlage für die Objektverwaltung sein.

Allerdings müßten zu diesem Zweck die relationalen Systeme um ein objektorientiertes Datenmodell erweitert werden. Hierbei könnten Datentabellen jeweils die Objekte einer Klasse speichern, während Vererbungs- und Methodeneigenschaften auf andere Weise eingerichtet werden müßten.

Objektorientierte Programmierung

Die objektorientierte Programmierung geht zurück auf die Arbeit mit abstrakten Datentypen (ADT) Ende der 60er Jahre. Bei einem abstrakten Datentyp werden einzelne Operationen und Datenrepräsentationen in einem Modul zusammengefaßt, so daß der Zugriff auf die Daten nur über das Modul möglich ist.

Bei OOP wird die Struktur eines Programms ebenfalls aus abstrakten Daten statt aus Prozeduren aufgebaut. Dadurch wird eine wesentlich bessere Strukturierung erzielt, wodurch Anwendungen einfacher zu erzeugen und zu warten sind.

Zukunftsperspektiven

Wohin der Weg der Speicherung und Verwaltung von Objekten geht, zeigen im Datenbankbereich die Implementierungen von Ingres, Release 6.3, aber auch von Informix Software oder Empress.

Die in diesem Artikel angesprochenen Konzepte sind jedoch noch viel weitreichender. Sie laufen innerhalb eines mehrjährigen Forschungsprojektes („Postgres") an der Universität von Berkeley in Kalifornien. Aus heutiger Sicht ist unser System so wie Ingres Ende der siebziger Jahre, nämlich „slow and buggy". Über den kommerziellen Erfolg Postgres wird der Markt erst in den neunziger Jahren entscheiden. ■

Quick-Pascal und Turbo-Pascal – gleiche Brüder?

Günter Jürgensmeier

Es ist in der letzten Zeit viel über die objektorientierten Möglichkeiten von Quick-Pascal und Turbo-Pascal diskutiert worden. Bei vielen Anwendern herrscht über die technischen Möglichkeiten nach wie vor beträchtliche Unkenntnis.

So wurde beispielsweise behauptet, daß Quick-Pascal keine virtuellen Methoden enthalte, während es tatsächlich so ist, daß in Quick-Pascal alle Methoden virtuell sind. In einigen Artikeln wurde das frühe und späte Binden („early binding"/„late binding") besprochen, und es wurde dort behauptet, daß das späte Binden zur Laufzeit irgendeine zeitaufwendige Suche notwendig macht – dem ist nicht so.

In diesem Artikel werden mehrere technische Aspekte besprochen, die beim Vergleich von Turbo-Pascal und Quick-Pascal wichtig sind. Die Absicht dieses Artikels ist es, Fehler zu korrigieren und klarzustellen, worin die Unterschiede zwischen den objektorientierten Erweiterungen dieser beiden Produkte bestehen.

Dynamische oder statische Objektanbindung

Dies ist sehr einfach; es entspricht völlig der Deklaration von Variablen oder Zeigern auf Variablen. Bei der traditionellen Programmierung ist es besser, ein Datum mit variabler Größe, ein sehr großes Datum oder eines mit unbestimmter Größe dynamisch anzulegen. Anstatt eine Variable zu deklarieren, deklariert man einen Zeiger auf eine Variable und greift darüber auf die Variable zu, wenn erforderlich.

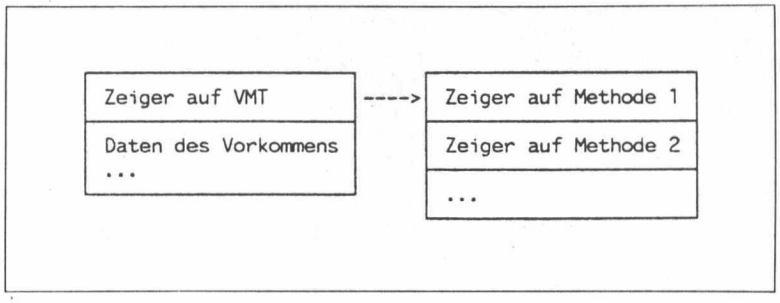
Bei der objektorientierten Programmierung (OOP) kann der Platz für die Daten eines Objekts statisch im Daten- oder Stack-Segment angelegt werden (statische Allokierung), oder man kann das Objekt dynamisch auf dem Heap anlegen (dynamische Allokierung). Es ist zu beachten, daß hier die Terminologie etwas irreführend ist, da „statische" Objekte, die lokal in einer Prozedur deklariert werden, nicht in demselben Sinn „statisch" sind, wie es globale Variablen sind. Wie lokale Variablen werden lokale „statische" Objekte beim Eintritt in eine Prozedur auf dem Stack angelegt und beim Verlassen der Prozedur wieder freigegeben.

Der Vorteil dynamischer Objekte besteht darin, daß sie keinen Platz im Daten- oder Stack-Segment benötigen (dort ist sowieso nur begrenzt Platz), und sie können nach Bedarf angelegt und freigegeben werden. Der Nachteil dynamischer Objekte besteht darin, daß sie beim Zugriff immer über ihren Zeiger angesprochen werden müssen, sie sind also langsamer.

Auf statische Objekte kann man schneller zugreifen, sie belegen jedoch Speicher im Stack- und Datensegment. Es ist darüber hinaus sehr wahrscheinlich, daß gut strukturierte Programme, die ausgiebig Prozeduren und Funktionen verwenden, die Vorteile statischer Objekte verringern werden. (Sobald ein statisches Objekt als ein Argument einer Prozedur übergeben wird, wird es im Prinzip, besonders aus Performance-Sicht, ein dynamisches Objekt.)

Virtuelle Methoden

C++ und Turbo-Pascal, Version 5.5 erlauben sowohl virtuelle als auch nichtvirtuelle Methoden; in Quick-Pascal sind alle Methoden virtuell. Virtuelle Methoden sind Methoden, auf die indirekt über eine „virtuelle Methodentabelle" (eine Sprungtabelle) zugegriffen wird. Ein Objekt, das virtuelle Methoden verwendet, sieht im Speicher so aus:



Die Struktur der VMT wird von der verwendeten Klassenhierarchie bestimmt. Ein Beispiel in Quick-Pascal:

```
{ QP Code }
type
        shape = object
                        procedure draw;
                        procedure erase;
                end;

        cube = object(shape)
                        procedure draw; override;
                        procedure rotate;
                end;
```

Der Autor Günter Jürgensmeier arbeitet als freier Journalist und Fachbuchautor in München. Seit 1989 ist er außerdem als Geschäftsführer des Synergy Verlages (München) tätig.

Methodentabelle

Die virtuelle Methodentabelle für „shape" hat zwei Einträge: einen Zeiger auf den Code der Prozedur „draw" und einen Zeiger auf den Code der Prozedur „erase". Die virtuelle Methodentabelle für „cube" hat drei Einträge: einen für „draw", einen für „erase" und einen für „rotate" (in dieser Reihenfolge).

In „cube" zeigt der Eintrag für „erase" auf genau den gleichen Programmcode, wie in der Tabelle für „shape", da dieser Code übernommen wurde (inherited). Der Eintrag für „draw" zeigt auf neuen Programmcode, da dieser übergangen wurde (override). Der Eintrag für „rotate" besteht ebenfalls aus neuem Code, und zwar deshalb, weil es sich hier um eine hinzugefügte Methode handelt.

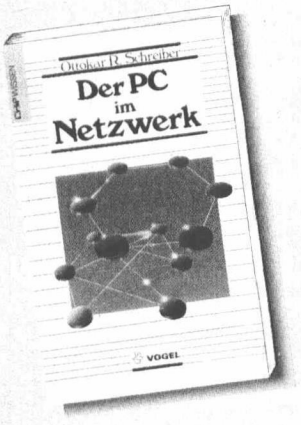
Die Struktur der VMT wurde übernommen (inherited). Die beiden ersten Einträge von „cube" (und jeder Unterklasse von „shape" oder „cube") sind immer Zeiger auf die Prozeduren „draw" und „erase". Der Vorteil davon ist, daß man dies schreiben kann:

```
var
        s : shape;
        c : cube;

        procedure ReDrawAThing( o : shape );
        begin
                o.Erase;
                o.Draw;
        end;

begin
        ...
        ReDrawAThing( s );
        ReDrawAThing( c );
        ...
end;
```

Da auf virtuelle Methoden immer indirekt über die VMT zugegriffen wird, ruft die Prozedur ReDrawAThing immer den „richtigen" „draw"- und „erase"-Code auf. Wenn im obigen Beispiel „draw" und „erase" keine virtuellen Methoden wären, würde der Code von ReDrawAThing die Tatsache ignorieren, daß „cube" die



CHIP WISSEN

Die kompetente Buchreihe rund um den PC

Ottokar R. Schreiber:

Der PC im Netzwerk

224 Seiten, 72 Bilder, 18 Tabellen
Hardcover, Format 24 x 16
40,– DM/ISBN 3-8023-**0262**-1

Dieses Buch bietet fachliches Grundwissen über die Vernetzung und stellt schwierige Sachverhalte verständlich dar. Umfassende Informationssysteme, zunächst als Insellösungen in Form lokaler Netzwerke, halten Einzug in Büros, Unis usw. Es entsteht die Forderung über Verbindung dieser Kommunikationsinseln über Gateways, mit Großrechnern sowie Daten- und Wissensbanken. Zahlreiche LANs verbinden sich zu einem Internet, in dem jeder Teilnehmer auf alle darin verfügbaren Ressourcen zugreifen kann. Also insgesamt ein interessantes Gebiet. Greifen Sie zu!!

Hans-Ulrich Wolff:

Schneller erfolgreich mit Lotus Manuscript 2.0

192 Seiten, 82 Bilder
Hardcover, Format 24 x 16
40,– DM/ISBN 3-8023-**0222**-2

Dieses Buch gibt eine systematische Einführung in eine leistungsfähige Dokumentationsverarbeitung. Es beschreibt in einer systematischen Einführung die Programmbefehle beispielhaft Schritt für Schritt. Die detaillierte Gliederung sowie das umfangreiche Stichwortverzeichnis ermöglichen dem engagierten und geübteren Anwender das rasche Nachschlagen der ihn speziell interessierenden Abschnitte. Das Erzeugen hochwertiger Ausdrucke mit Grafiken, Tabellen und Gleichungen macht seinen Einsatz vor allem für technisch-wissenschaftliche Anwendungen interessant.

VOGEL

Haben Sie schon den neuen „CHIP- Katalog" ?
Bestellen Sie gleich!

Vogel Buchverlag
Postfach 67 40 · 8700 Würzburg 1

„draw"-Methode übergangen hat, und immer den „draw"-Code von „shape" aufrufen, da der Parameter „o" als Objekt vom Typ „shape" deklariert wurde.

Die Vorteile virtueller Methoden bestehen in der wesentlich besseren Wiederverwendbarkeit von Code und im wesentlich logischeren Verhalten. Im obigen Beispiel macht es Sinn, daß der neue Methodencode aufgerufen wird, da der Code für „draw" in der „cube"-Klasse übergangen wurde. Die ausgiebige Verwendung virtueller Methoden ist in gut entworfenem, objektorientiertem Code häufig.

Der Nachteil virtueller Methoden ist der langsamere Zugriff. Nicht virtuelle Methoden werden in einfache Aufrufe ohne Indirektion aufgelöst.

Konstruktoren und Destruktoren

Dies sind besondere Methoden, die bei der Initialisierung und Beendigung eines Objekts behilflich sind. In C++ sind die Konstruktoren und Destruktoren sehr leistungsfähige Bestandteile. Der Compiler ruft die Konstruktoren und Destruktoren automatisch auf, wenn Objekte deklariert und wenn sie beendet werden. Bei statischen Objekten in einer Prozedur (und als solche werden sie auf dem Stack angelegt) kann dies sehr angenehm sein, da C++ die Destruktoren von Objekten aufruft, sobald die Objekte ihren Gültigkeitsbereich verlassen (etwa wenn die Prozedur verlassen wird). Dieser automatische Aufruf ist ein wichtiger Aspekt der Konstruktoren und Destruktoren in C++.

Turbo-Pascal 5.5 verwendet die Konstruktoren-/Destruktoren-Terminologie, implementiert in Wirklichkeit jedoch die Funktionalität von C++ nicht. In Turbo-Pascal müssen Konstruktoren explizit aufgerufen werden und unterscheiden sich nur wenig von normalen Methoden. Die beiden einzigen Eigenheiten der Konstruktoren in Turbo-Pascal 5.5 sind:

1. Man muß einen Konstruktor aufrufen, um die virtuelle Methodentabelle für ein Objekt einzurichten, das virtuelle Methoden verwendet.
2. Konstruktoren können als Teil von „new" aufgerufen werden:

```
{ TP 5.5 code }
type
        point = object
                    x, y : integer;
                    constructor init;
                    procedure draw; virtual;
                end;
var
        pp : ^point;
begin
        new( pp, init);
end;
```

Die Destruktoren von Turbo-Pascal 5.5 sind im Vergleich zu C++ ähnlich eingeschränkt. In Turbo-Pascal 5.5 besteht der einzige Unterschied zwischen einer Destruktor-Methode und einer normalen Methode darin, daß ein Destruktor bei „dispose" aufgerufen werden kann. Hier ein kurzes Beispiel:

```
{ TP 5.5 code }
type
        point = object
                    x, y : integer;
                    constructor init;
                    destructor done;
                    procedure draw; virtual;
                end;
        point3d = object(point)
                    z : integer;
                    destructor done;
                end;
var
        pp :  ^point;
        p3p : ^point3d;

procedure goaway( var p : point);
begin
        dispose( p, done);
end;

begin
        new( pp, init);
        new( p3p, init);
        goaway( pp );
        goaway( p3p );
end;
```

Das Besondere an den Destruktoren von Turbo-Pascal 5.5 besteht darin, daß sie, wenn sie als Teil von „dispose" aufgerufen werden, die Anzahl der freizugebenden Byte übergeben. Im obigen Beispiel gibt die Prozedur „goaway" die richtige Anzahl Byte frei, da die Destruktoren dem „dispose"-Aufruf diese Information übergeben.

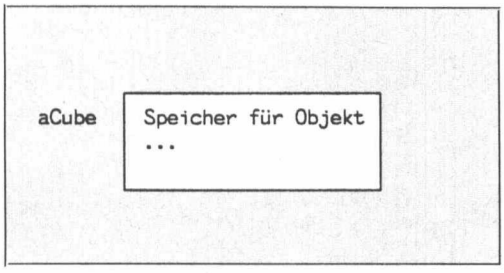
Quick-Pascal verfügt nicht über Konstruktoren und Destruktoren, da es sie nicht braucht. Quick-Pascal richtet die virtuelle Methodentabelle automatisch ein, wenn ein Objekt angelegt wird. Der „dispose"-Aufruf in Quick-Pascal gibt automatisch die richtige Anzahl Byte frei, wenn ein Objekt freigegeben wird (sogar in einer Prozedur wie „goaway", in der das Objekt als Parent deklariert wurde). Der „dispose"-Aufruf weiß die richtige Anzahl Byte, die freigegeben werden müssen, da diese Information als Teil der virtuellen Methodentabelle gespeichert wird.

Binde-Strategie und Performance

Als „Binden" wird bezeichnet, wie ein Compiler einen Methodenaufruf in wirkliche Maschinenbefehle umsetzt, die den entsprechenden Code aufrufen. Das Binden

wird von zwei Dingen beeinflußt: statische und dynamische Objektanbindung und virtuelle oder nicht-virtuelle Methoden. Wir wollen zunächst untersuchen, wie auf die Daten eines Objekts zugegriffen wird.

Bei der statischen Objektanbindung werden Objekte im Daten- oder Stack-Segment angelegt, abhängig davon, ob es sich um globale oder lokale Objekte handelt:



Im Fall eines globalen statischen Objekts ist der Name des Objekts eine direkte Speicherreferenz. Der Zugriff auf die Daten des Objekts ist ein direkter Speicherzugriff, der keine Indirektion erfordert. In Assembler würde das etwa so aussehen:

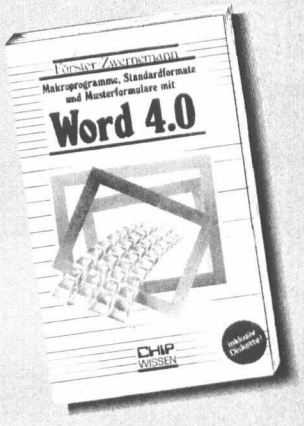
```
MOV     AX, Datenadresse
```

Dies ist identisch mit dem Zugriff auf globale Variablen. Der Zugriff auf Daten bei einem dynamischen Objekt erfordert eine Ebene der Indirektion:

Um etwa auf eine Vorkommen-Variable eines dynamischen Objekts zuzugreifen, muß in Assembler etwa folgendes gemacht werden:

```
LES   DI, aCube
MOV   AX, ES:[DI+Datenoffset]
```

Es ist zu beachten, daß in beiden Fällen, sowohl bei statischen als auch bei dynamischen Objekten, zur Laufzeit keine Suche erforderlich ist. In beiden Fällen kann der Compiler beim Kompilieren alle Infor-



CHIP WISSEN

Die kompetente Buchreihe rund um den PC

Förster/Zwernemann:

Makroprogramme, Standardformate und Musterformulare mit Word 4.0

144 Seiten, 42 Bilder, Hardcover
mit 5,25-Zoll-Diskette
48,– DM/ISBN 3-8023-**0246**-X

Jeder der mit Word 4.0 Standardformulare ausfüllt, Serientexte schreibt und Etiketten bedruckt oder eigene Makroprogramme und Druckformatvorlagen erstellen möchte, braucht dieses Buch mit Diskette.

Auf der Diskette befinden sich nahezu alle Etikettenformate, Formate für Endlospapiere und Standardformulare.
Die Makroprogrammierung, das Arbeiten mit Druckformaten und Serientexten wird Schritt für Schritt anhand von Praxis-Beispielen erklärt.

Rationelle Textverarbeitung mit WordPerfect 5.0

Textbausteine, Makros, Formulare, Grafiken
ca. 160 Seiten, 38 Bilder, Hardcover
mit zwei 5,25-Zoll-Disketten
48,– DM/ISBN 3-8023-**0295**-8

Wie vielfältig das Textverarbeitungsprogramm WordPerfect einsetzbar ist, zeigt dieses Buch dem fortgeschrittenen Benutzer auf.

Ob Sie Ihre Korrespondenz mit Textbausteinen und Serienbriefen rationalisieren, die Tastaturbedienung vereinfachen, Funktionen mit Makros automatisieren, Formulare einschließlich Rechenfunktionen und Geschäftsgrafiken erstellen wollen: Hier finden Sie eine Fülle von Anwendungsmöglichkeiten und Anregungen aus und für die Büropraxis, unterstützt durch zahlreiche Beispieldaten auf den beiden Disketten. Damit eignet sich das gut strukturierte Werk auch als Begleitmaterial für WordPerfect-Schulungen.

VOGEL

Haben Sie schon den neuen „CHIP- Katalog" ?
Bestellen Sie gleich!

Vogel Buchverlag
Postfach 67 40 · 8700 Würzburg 1

mationen feststellen, die er wissen muß: bei statischen Objekten die gewünschte Datenadresse, bei dynamischen Objekten den Offset der Daten im Speicher des Objekts.

Einflüsse

Der Aufruf einer Methode eines Objekts wird davon beeinflußt, ob die Methode virtuell ist oder nicht. Wenn die Methode virtuell ist, muß die Adresse der Methode aus der Methoden-Sprungtabelle herausgesucht werden.

Für den Aufruf einer virtuellen Methode bei einem dynamischen Objekt ist folgendes notwendig.

Es ist zu beachten, daß selbst bei virtuellen Methoden und dynamischen Objekten zur Laufzeit keine „Suche" notwendig ist. Bereits bei der Kompilierung kann der Compiler den Offset der gewünschten Methode feststellen und den entsprechenden Code erzeugen.

Beim Aufruf einer virtuellen Methode bei einem statischen Objekt gibt es einfach nur eine Ebene der Indirektion weniger (Unter der Voraussetzung, daß sich das zu bearbeitende statische Objekt im Datensegment befindet).

Keine Probleme

Der Aufruf einer nicht virtuellen Methode bei einem statischen Objekt ist einfach. Der Aufruf wird zu einem einfachen Prozeduraufruf.

Ganz ohne Frage beeinflussen die Objektanbindungs-Strategie (statisch oder dynamisch) und die Methodenzugriffs-Strategie (virtuell oder nicht) die Performance des kompilierten Codes. Da leistungsfähige Anwendungen dazu tendieren, dynamische Anbindung und virtuelle Methoden zu verwenden, müßte die Performance objektorientierter Anwendungen theoretisch schlechter sein als die traditioneller Anwendungen.

Performance zweitrangig

In der Praxis ist die Performance jedoch nur selten ein Thema. C++ und objektorientierte Erweiterungen von Pascal bieten eine gute Mischung traditioneller und objektorientierter Techniken. Dadurch kann der Benutzer eine klare Trennlinie ziehen, wo objektorientierte und wo traditionelle Techniken eingesetzt werden. Typischerweise werden objektorientierte Techniken für die Erledigung von Dingen auf übergeordneter Ebene eingesetzt, wie:

- Benutzerschnittstellen,
- Gerätesimulation,
- komplexe Datenstrukturen,
- grafische Objekte.

Diese Dinge arbeiten zumeist nur mit „Benutzergeschwindigkeit". (Die Anlage eines Fensters braucht nur so schnell zu arbeiten, wie der Benutzer Fenster anlegen kann, das heißt für einen Computer nicht besonders schnell.)

Die traditionellen Techniken werden für systemnähere Operationen verwendet, die einen so geringen Overhead wie möglich erfordern.

Die Performance von C++ und objektorientierten Pascal-Versionen sollte nur mit „reinen" objektorientierten Systemen wie Smalltalk und Actor verglichen werden, in denen selbst Daten auf niedriger Ebene (Konstanten, Integer, Strings, Arrays usw.) als Objekte implementiert sind.

Zusammenfassung

Turbo-Pascal 5.5 bietet einige Dinge, die Quick-Pascal 1.0 nicht aufzuweisen hat. Die wichtigsten sind:

1. statisch angelegte Objekte und
2. nicht-virtuelle Methoden.

Dadurch werden Performance-Vorteile auf Kosten anderer Dinge erzielt: Verlust von Daten- oder Stack-Speicher im Fall statisch angelegter Objekte oder Verlust der Wiederverwendbarkeit von Code im Fall der nicht-virtuellen Methoden.

In der Praxis sieht es so aus, daß nicht triviale, objektorientierte Programme zumeist virtuelle Methoden deklarieren und viele dynamisch angelegte Objekte verwenden. Die Quick-Pascal-Programmierumgebung wurde in Quick-Pascal geschrieben und macht ausgiebigen Gebrauch von seinen objektorientierten Möglichkeiten. Die Performance dynamischer Objekte und virtueller Methoden war also ein wichtiges Thema bei der Entwicklung.

Der Einsatz von Konstruktoren und Destruktoren in Turbo-Pascal 5.5 ist schlecht gelöst. Erstens wird zwar die Terminologie von C++ verwendet, die Implementierung ist jedoch wesentlich weniger leistungsfähig. Insbesondere das Fehlen des automatischen Aufrufs bei der Anlage eines Objekts und das Löschen (besonders bei statischen Objekten) sind enttäuschend. Zum zweiten scheint die Notwendigkeit von Konstruktoren und Destruktoren in Turbo-Pascal ein Ergebnis einer schwachen Implementierung zu sein, weniger der Wunsch, neue Funktionalität zu bieten. Turbo-Pascal 5.5 benötigt Konstruktoren und Destruktoren, um:

1. die virtuelle Methodentabelle einzurichten und
2. den „dispose"-Aufruf über die Objektgröße zu informieren.

Quick-Pascal erzielt diese beiden Punkte ohne zusätzliche konzeptionelle Belastungen. Virtuelle Methodentabellen werden automatisch bei der Anbindung eines Objekts eingerichtet. Die „dispose"-Funktion weiß automatisch die richtige Größe eines Objekts. ■

Können herkömmliche Programmiersprachen überleben?

Silvia Gutschmidt

Die Autorin Silvia Gutschmidt studierte an der Universität Regensburg Volkswirtschaftslehre. Seit 1985 arbeitet sie als freie DV-Journalistin, Autorin und DV-Beraterin in Eichenau bei München.

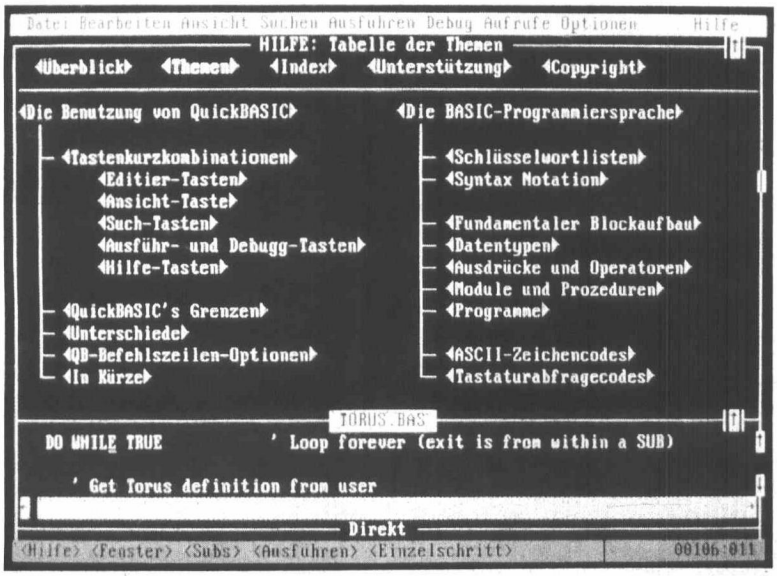
Angesichts des beginnenden Siegeszuges der objektorientierten Programmiersprachen stellt sich die Frage, wie und ob die etablierten Programmiersprachen überleben. BASIC etwa ist ein Programmierwerkzeug für Tüftler und Hobby-Programmierer, aber keines für professionelle Software-Entwickler. Im folgenden sollen die Funktionalität und der Markt von BASIC näher untersucht werden.

Um Klarheit über dieses Marktsegment zu erhalten, hat der Software-Hersteller Microsoft eine Untersuchung gestartet. Dieser Studie zufolge schreiben 62 Prozent der BASIC-Programmierer Software für Drittanwender. 50 Prozent der Quick-BASIC-Anwender schreiben Software, die für den Verkauf bestimmt ist. Innerhalb einer 25jährigen Entwicklung wurden in BASIC eine Vielzahl von Eigenschaften moderner Programmiersprachen integriert.

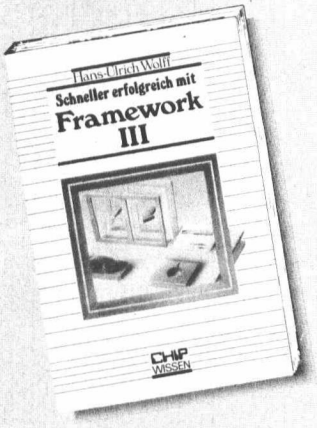
Zwar ist die Zahl der BASIC-Programmierer mit der Präsentation höher entwickelter Programmiersprachen wie etwa C leicht zurückgegangen. Jedoch haben zu BASIC mehr Anwender Zugriff als zu irgendeiner anderen Sprache. Das einfache Erlernen und Anwenden sind wichtige Faktoren und werden deshalb die Stärken von BASIC bleiben. Die Fähigkeit von BASIC, im Funktionsumfang und der Leistung zu wachsen, während der Vorzug der einfachen Einsetzbarkeit gewahrt bleibt, läßt BASIC gute Chancen, viele andere Programmiersprachen zu überleben. Pascal und C haben unter professionellen Programmierern in den letzten Jahren zwar an Boden gewonnen, BASIC hat seine Position dennoch gehalten.

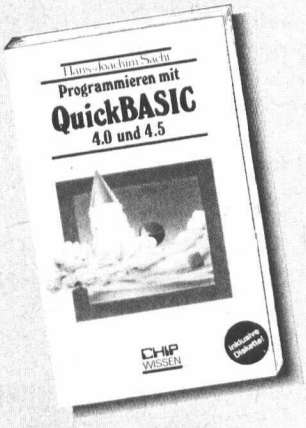
BASIC wird oft vorgeworfen, daß die Debugging-Werkzeuge ungeeignet für große komplexe Programme seien. BASIC umfaßt jedoch alle Debugging-Werkzeuge, die auch bei anderen professionellen Programmiersprachen zu finden sind. Besonders für professionelle Programmierer bietet der Microsoft BASIC Compiler 6.0 einen ausgereiften Editor, den Microsoft Code View Debugger, einen Inkremental-Linker und weitere technisch fortschrittliche Werkzeuge. Diese Werkzeuge sind identisch mit denen, die Microsoft bei der Entwicklung von Applikationen wie Microsoft Excel oder Microsoft Word benutzt. Die integrierte Programmierung von Microsoft Quick-BASIC bietet eine Debugging-Leistung, die kaum von einer anderen professionellen Programmiersprache übertroffen wird. So minimiert die Syntaxprüfung beispielsweise typ- und syntaxbezogene Fehler. Das Direktfenster gestattet dem Anwender, den Wert von Variablen während des Programmablaufs zu ändern oder Teile des Programms im Ablauf zu testen. Programmierer haben die Möglichkeit, ihr Programm zu edieren und ohne Neustart laufenzulassen. Andere Funktionen umfassen das Setzen von Haltepunkten, Anzeigenvariablen und konditionalen Stoppbedingungen. Keine andere Programmiersprache bietet ein Quellcode-Debugging wie der interaktive BASIC p-code-Sprachkern, der das Kernstück von Microsoft Quick-BASIC bildet.

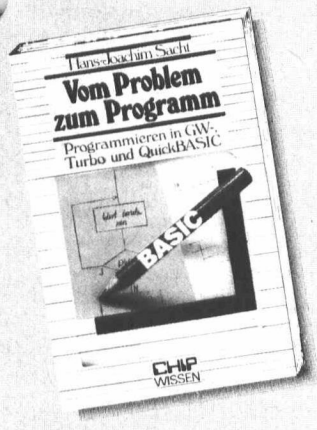
Auch die Behauptung, mit BASIC erreiche man keinen dichten und schnellen Code, wie er für Hochleistungs-Tasks erforderlich ist, stimmt nicht. Als hochoptimierender Compiler erzeugt der Microsoft BASIC-Compiler einen sehr effizienten Code und enthält verschiedene Optimierungstechniken. Programmierer haben darüber hinaus die Möglichkeit, Assembler-Subroutinen aufzurufen, falls spezielle Anforderungen es nötig machen. Die BASIC-Runtime-Bibliothek ist speziell auf hohe Geschwindigkeit abgestimmt. Zahlreiche Benchmarks haben gezeigt, daß Microsoft Quick-BASIC weitverbreitete Programmiersprachen wie Turbo-Pascal übertrifft. ▶



Der Quick-BASIC-Ratgeber hält Informationen zur Quick-BASIC-Entwicklungsumgebung bereit







CHIP WISSEN

Die kompetente Buchreihe rund um den PC

Honerkamp/Jetter:

Fliegen mit dem Flight Simulator III

244 Seiten, zahlr. Bilder, Harteinband,
mit 5,25-Zoll-Mode-Diskette
40,– DM/ISBN 3-8023-**0274**-5

Das Buch beschreibt ausführlich das Flieger-Einmaleins, das zum Beherrschen des Simulationsprogramms **„Flight Simulator III"** notwendig ist. Ausführlich wird die Instrumentenfliegerei mit den dazugehörigen Karten behandelt. Einige Neuerungen, wie der Anflug auf San Francisco Area und Scenery Disk Europe, der Anflüge auf bekannte deutsche und europäische Flughäfen erlaubt, erhöhen den Spaß an diesem Buch.
Die Autoren wünschen schon heute „Hals- und Beinbruch" und „happy landings".

Hans-Ulrich Wolff:

Schneller erfolgreich mit Framework III

180 Seiten, 55 Bilder
Harteinband, Fadenheftung
38,– DM/ISBN 3-8023-**0287**-7

Das Buch beschreibt die deutsche Framework-III-Version. Es erleichtert den Einstieg und unterstützt den Anwender beim Bewältigen der vielfältigsten Aufgaben anhand leicht nachvollziehbarer Beispiele. Die Programmiersprache FRED wird einführend und ihre Funktion umfassend vorgestellt.

Framework III eignet sich hervorragend für die Bewältigung der typischen Aufgabenstellungen am MS-DOS-Rechner:
● konzeptionelles Arbeiten ● Textverarbeitung ● Tabellenkalkulation ● Datenbanken ● Grafik und Telekommunikation

Hans-Joachim Sacht:

Programmieren mit QuickBASIC 4.0 und 4.5

210 Seiten, 66 Bilder, Harteinband
mit 5,25-Zoll-Diskette
48,– DM/ISBN 3-8023-**0245**-1

Durch die neuen Microsoft-Compiler QuickBASIC 4.0 und 4.5 können so komfortable und gutstrukturierte Programme entwickelt werden, wie es sonst nur mit einem leistungsfähigen Interpreter möglich war.

Das Buch bietet die vielfältigen Funktionen in konzentrierter Form und geht vor allem auf die Handhabung und Sprachelemente dieses Compilers ein. Programmierern, die bisher nur mit einem BASIC-Interpreter gearbeitet haben, wird erklärt, wie sie schnell auf QuickBASIC umsteigen und so die Vorteile der Kompilierung ausnutzen können. Eine Diskette mit den im Buch behandelten Beispielen liegt bei.

Hans-Joachim Sacht:

Vom Problem zum Programm

Programmieren in GW-BASIC, Turbo BASIC und QuickBASIC

464 Seiten, 183 Bilder
4., völlig überarbeitete Auflage 1988
48,– DM/ISBN 3-8023-**0214**-1

Das von Microsoft entwickelte GW-BASIC bzw. BASICA gilt inzwischen bei 16-bit-Computern als Standard und für 32-bit-Rechner als Grundlage des Sprachumfangs auch anderer neuer Dialekte.
Die Neukonzeption gliedert sich in 3 Teile:

1. Teil erklärt Grundlagen der BASIC-Programmierung unter Einbeziehung der wichtigsten MS-DOS-Kommandos, Compiler.
2. Teil bringt zahlreiche Programmroutinen für breite Anwendungen.
3. Teil zeigt, wie ein Programm Schritt für Schritt entsteht.

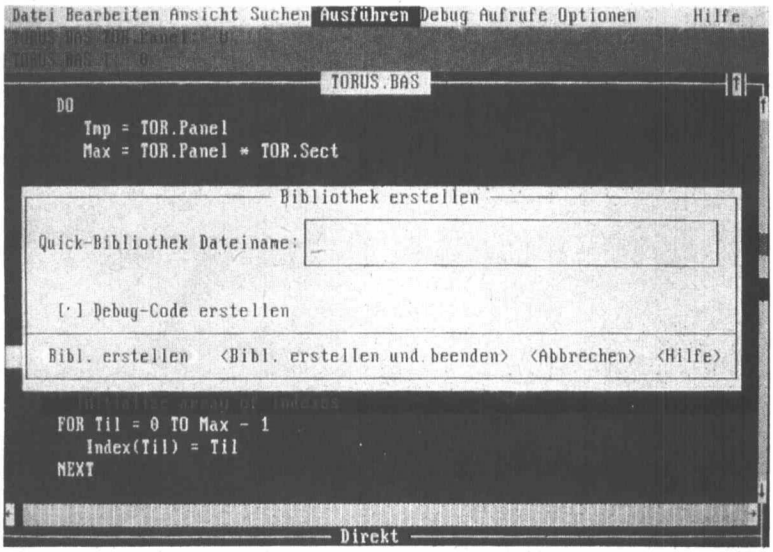
Eine Diskette mit den im Buch behandelten Beispielen ist erhältlich.

VOGEL

Haben Sie schon den neuen „CHIP- Katalog" ?
Bestellen Sie gleich!

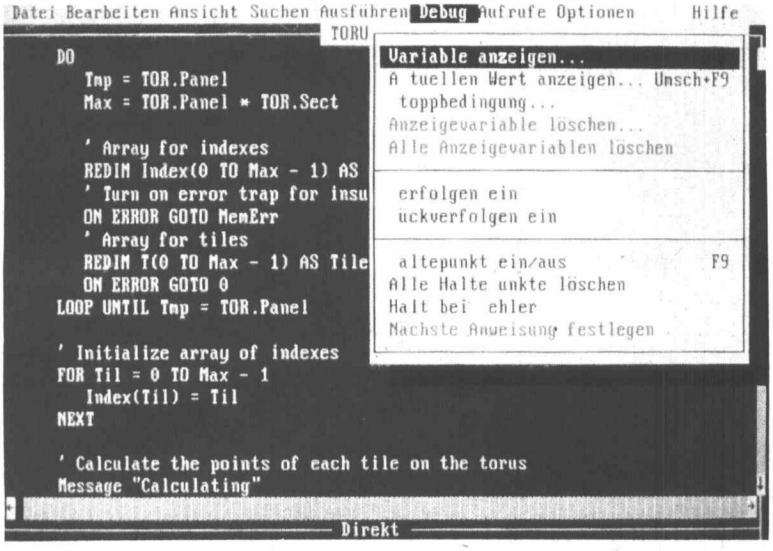
Vogel Buchverlag
Postfach 67 40 · 8700 Würzburg 1

Anlegen einer Bibliothek in Microsoft Quick-BASIC



Es ist auch kein Problem, in BASIC einen sauberen Code zu schreiben. Die heutigen BASIC-Compiler sind weit entfernt von GW-BASIC, BASIC A und anderen früheren Compilern. Mit modernen BASIC-Compilern können BASIC-Programme genauso strukturiert geschrieben werden wie in anderen Programmiersprachen auch. BASIC ist ebenso einfach oder noch einfacher anzuwenden als andere strukturorientierte Programmiersprachen.

Selbst Großunternehmen können problemlos BASIC einsetzen, denn Microsoft hält nahezu 99 Prozent Marktanteil bei PC-BASIC. Damit ist diese Sprache wie keine andere standardisiert. Darüber hinaus ist Microsoft BASIC für nahezu alle Industriestandard-Mikrocomputer, auf Apple-, IBM-DOS-, MS OS/2-, Xenix- und CP/M-Betriebssystem verfügbar. Nachdem es BASIC-Implementierungen auf einer breiten Palette von Computern – von Mikros bis zu Großcomputern – gibt, ist die Portierung von BASIC-Programmen auf andere Umgebungen einfach und weit verbreitet.



Das Debug-Menü ist aufgerufen

Ist BASIC überflüssig?

Interessant wird es bei der Frage der objektorientierten Sprachen, die allgemeinhin als die Zukunft in der Programmierung gelten und BASIC so quasi ad absurdum führten. Hier ist anzumerken, daß, sobald die objektorientierte Programmierung zur Standardmethode wird, Microsoft auch ein objektorientiertes BASIC zur Verfügung stellen will – so wie das Unternehmen seinerzeit die Umstellung von der linearen zur blockstrukturierten Programmierung vorgenommen hat, als erwiesen war, daß die blockstrukturierte Programmierung die bessere Lösung ist.

BASIC ist auch in der Lage, die komplexen Funktionen der MS-OS/2-Umgebung zu unterstützen. Der in der Bundesrepublik im März 1988 eingeführte Microsoft BASIC Compiler 6.0 unterstützt als erster BASIC-Compiler MS OS/2 und wird in Zukunft auch den MS OS/2 Presentation Manager unterstützen.

Breites Einsatzspektrum

BASIC ist nicht, wie vielfach angenommen, nur eine Einzelanwender-Programmiersprache – es ist auch in der Lage, große gruppenorientierte Entwicklungsprojekte zu unterstützen, in denen modulare Programmierung und Code-Sharing üblich sind. Denn das heutige BASIC ist eine hochgradig strukturierte Programmiersprache, die in starkem Maße Top-down-Entwicklungen, Modularität und andere gruppenorientierte Methoden unterstützt. Da BASIC einfach zu lesen und zu verstehen ist, können neue Programmierer leicht in ein laufendes Projekt integriert werden und den bislang erzeugten Code sofort verstehen.

Während Pascal typischerweise eingesetzt wird, um Informatikstudenten mit dem Programmieren vertraut zu machen, wird BASIC in allen anderen Bereichen bevorzugt. BASIC hat nun einmal die gleichen Daten- und Programmstrukturen wie Pascal, so daß viele Schulen auch BASIC in ihr Informatik-Lehrangebot aufnehmen.

Markt

Bei Microsoft wird die Meinung vertreten, daß der BASIC-Markt auch weiterhin stark sein werde. Das Unternehmen hat seit 1985 weltweit 500 000 Quick-BASIC-Pakete verkauft. Quick-BASIC ist die meistverkaufte Microsoft-Programmiersprache in Europa und Japan. Ferner verspricht die zunehmende Bedeutung programmierbarer Applikationen auch weiterhin ein lukratives Marktpotential für diese „herkömmliche" Programmiersprache. ■

Datenbank mit Objektverwaltung und Regelsystem

Ursula Heitfeld

Im Zuge seiner Strategie, Technologien für das unternehmensweite integrierte Informations-Management bereitzustellen, hat das Unternehmen Ingres sein gleichnamiges Datenbanksystem mit neuartigen Leistungsmerkmalen ausgestattet.

Neben einem funktional erweiterten Datenbank-Manager stehen nun ein Object Manager für objektorientiertes Arbeiten und ein Knowledge Manager für den Aufbau regelbasierter Systeme zur Verfügung. Anwender sind damit in der Lage, eine auf die Erfordernisse ihrer Branche beziehungsweise ihres Unternehmens zugeschnittene Datenbankapplikation zu erarbeiten. Da nicht nur Daten, sondern auch Regeln und Objekte verwaltet werden können, ist es möglich geworden, in einer Datenbank Anforderungen, Richtlinien und Vorgehensweisen der täglichen Geschäftspraxis festzuhalten.

Der Knowledge Manager für regelbasierte Systeme

Der Knowledge Manager verarbeitet nicht nur Daten, sondern auch Regeln. Unternehmen sind damit in der Lage, das Wissen oder die Regeln, die die EDV eines Projekts, einer Abteilung oder eines Unternehmens bestimmen, in der Datenbank festzuschreiben. Die Einhaltung der Regeln wird automatisch vom System überwacht.

Bei einem Lagerhaltungssystem könnten etwa die Mindestlagermengen, die Dispositionsrichtlinien und Bestellmengen als ein Regelwert erzeugt werden. Mußte früher der Anwendungsprogrammierer dafür Sorge tragen, daß jede Anwendung den Unternehmensregeln genügt, so trägt heute der Knowledge Manager diesen Aufgaben Rechnung. Nachdem der Datenbankadministrator die Regeln definiert hat, achtet das System darauf, daß die Regeln eingehalten werden. Das bewirkt nicht nur eine erhebliche Produktivitätssteigerung, sondern verschafft einem Unternehmen auch mehr Flexibilität. Sollten sich zum Beispiel die Dispositionsrichtlinien ändern, wird die entsprechende Regel modifiziert. Sie besitzt dann für diese Unternehmensdatenbank und dadurch automatisch für alle damit arbeitenden Anwendungen Gültigkeit.

Als ein weiteres Beispiel für eine Regel kann die referentielle Integrität herangezogen werden. Mit dem Knowledge Manager lassen sich beliebige Bezüge zwischen Tabellen definieren, die automatische Prüfungen bei Wertveränderungen, Einfügungen oder Löschungen von Daten veranlassen. Damit steht dem Anwender ein komplettes Instrumentarium im Rahmen des Regelsystems zur Verfügung, die Anforderungen an referentielle Integrität effektiv und flexibel zu realisieren. Die referentielle Integrität stellt somit eine Teilfunktionalität des Regelsystems von Ingres dar.

Die Anzahl der verwalteten Regeln, die als Datenbankprozeduren abgefaßt sind, ist beliebig. Die Regeln können sowohl rekursiv als auch verschachtelt aufgebaut sein. Das bedeutet, eine Regel kann sich selbst, aber auch andere Regeln aktivieren. Rekursion ist beispielsweise für den Einsatz im Rechnungs- oder Personalwesen nützlich. Als Auslöser für das Ansprechen einer Regel kommen Benutzerzugriffe (Einfügen, Löschen, Aktualisieren von Daten) ebenso in Frage wie Wertveränderungen der gespeicherten Daten.

Das regelbasierte System ist nicht mit bloßen Trigger-Mechanismen zu verwechseln, wie sie zum Teil in anderen Datenbanken zum Einsatz kommen. Trigger können keine Wertveränderungen registrieren und erlauben zudem keine oder nur eine eng begrenzte Verschachtelung von Regeln. Zudem arbeiten Trigger nicht rekursiv, so daß komplexe Trigger-Regeln nicht aufzustellen sind.

Der Nutzen des Knowledge Managers liegt jedoch gerade darin, daß sich mit ihm komplexe Regelwerke oder Wissensbasen erzeugen lassen und daß deren Einhaltung das System überwacht. Nur auf diese Weise läßt sich das Datenbanksystem optimal auf die jeweilige Branche beziehungsweise das Unternehmen anpassen.

Zum Knowledge Manager gehören weiterhin ein neuartiges Resource-Control-System und neue Mechanismen für die Zugriffsautorisierung.

Resource-Control-System

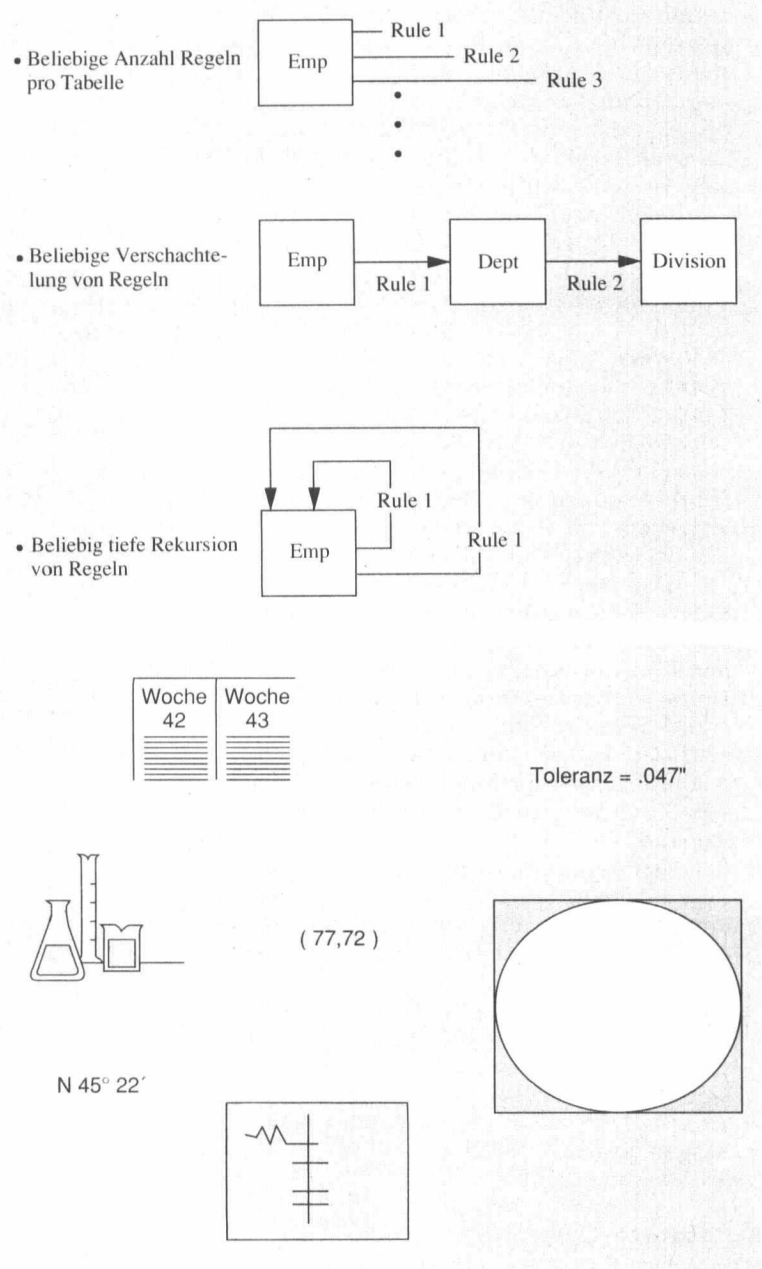
Das Resource-Control-System erlaubt es dem Systemadministrator, eine Obergrenze für den Ressourcenverbrauch von Datenbankabfragen festzulegen. Dadurch läßt sich verhindern, daß Benutzer versehentlich Abfragen starten, die die Ressourcen der jeweiligen Datenbankumgebung über Gebühr beanspruchen. Ohne die Möglichkeit zur Begrenzung könnte ein Benut-

Die Autorin Ursula Heitfeld ist als Marketing-Leiterin bei der Ingres GmbH in Frankfurt am Main tätig.

zer eine Abfrage eingeben, deren Ausführung möglicherweise Stunden dauert. Diese Gefahr ist insbesondere gegeben, wenn Endbenutzer Zugang zu großen Datenbeständen erhalten, was für viele Anwendungen wünschenswert oder notwendig ist.

Das Regelsystem stellt einen wichtigen Baustein für die „Intelligenz" von Ingres dar

Die Komplexitätsobergrenze kann vom Systemadministrator anhand zweier Kriterien festgesetzt werden: anhand der Anzahl der Plattenzugriffe für eine einzelne Abfrage und/oder der Anzahl der vom Systems. Der Optimizer arbeitet auf statistischer Basis. Dadurch wird verhindert, daß eine Abfrage zunächst beginnt und bereits beträchtliche Systemressourcen beansprucht, bevor sie wegen zu hoher Komplexität abgebrochen wird.



Ingres verarbeitet auch komplexe Objekte, mit denen ein herkömmliches Datenbanksystem nicht umgehen kann

stem ausgegebenen Zeilen einer Datentabelle. Eine Abfrage, die die festgelegte Grenze überschreiten würde, weist Ingres bereits vor der Ausführung zurück.

Die Grundlage für die vorausschauende Berechnung der Komplexität bildet der Query Optimizer des Datenbanksy-

Group, Application und Feature Permits

Mit Group, Application und Feature Permits führt Ingres eine signifikante Flexibilisierung im Bereich der Zugriffsautorisierung ein.

Das Leistungsmerkmal der Group Permits ermöglicht es dem Datenbankadministrator, Zugriffsautorisierungen benutzergruppenweise vorzunehmen. Dies erleichtert nicht nur seine Tätigkeit, sondern erhöht auch die Übersichtlichkeit bei der Systemadministration. So läßt sich etwa mit einem einzigen Kommando erreichen, daß alle Abteilungsleiter eines Unternehmens Zugriff auf bestimmte Datenbestände erhalten.

Durch Application Permits werden Zugangsrechte zu Datenbanken an Anwendungen statt an einzelne Benutzer gekoppelt. Auf diese Weise läßt sich erreichen, daß auf Daten ausschließlich über die dafür vorgesehenen Applikationsprogramme zugegriffen wird. Kontrolleinrichtungen in Anwendungs-Software lassen sich somit nicht durch direkte Datenbankzugriffe umgehen.

Die Identifizierung der Anwendungsprogramme erfolgt über Paßwörter, die verschlüsselt in der Datenbank abgelegt werden. Damit wird es auch für Hacker unmöglich, eine Datenbank durch Vortäuschen eines gültigen Applikationsprogramms zu „knacken". Durch Applikation Permits wird somit die Systemsicherheit beträchtlich erhöht.

Über Feature Permits können Berechtigungen für einzelne Leistungsmerkmale beim Umgang mit Datenbanken selektiv vergeben werden. So läßt sich beispielsweise die willkürliche Neuanlage von Tabellen oder Datenbankprozeduren verhindern.

Der Object Manager für objektorientierte Systeme

Das Produkt Object Manager erlaubt dem Benutzer die Definition neuer Datenobjekte. Hierbei kommen alle Arten von Objekten in Frage, beispielsweise für Artikelkodierungen, geografische Koordinaten, Temperaturen, Zeitserien etc. Der Anwender ist in der Objektdefinition völlig frei.

Die Objekte können in Ingres nicht nur gespeichert und abgerufen werden, sondern lassen sich mit Operatoren (+, −, *, >, <, = etc.) manipulieren. Damit können die benutzerdefinierten Objekte unein-

geschränkt in die Verarbeitung etwa mit SQL einbezogen werden. Der Code für die Unterstützung neuer Datenobjekte wird in der Programmiersprache C abgefaßt und in den Datenbank-Manager eingebunden. Dadurch wird erreicht, daß die Performance benutzerdefinierter Objekte der der Standarddatentypen gleichwertig ist. Die Grundlage hierfür stellt die Abstract Datatype Facility (ADF) dar, die Bestandteil der Ingres-Architektur ist.

Die mit ADF realisierte Objektunterstützung ist nicht zu verwechseln mit der bloßen Möglichkeit, Binärketten zu speichern und abzurufen, wie sie teilweise in anderen Datenbanksystemen gegeben ist. Die Ingres-Objektverwaltung ist ebenfalls nicht zu vergleichen mit Einrichtungen für benutzerdefinierte Datentypen. In diesen Fällen werden lediglich vorhandene Datentypen leicht modifiziert – beispielsweise durch Einschränkung des Wertebereichs – während Ingres die Definition völlig neuer Datenobjekte zuläßt.

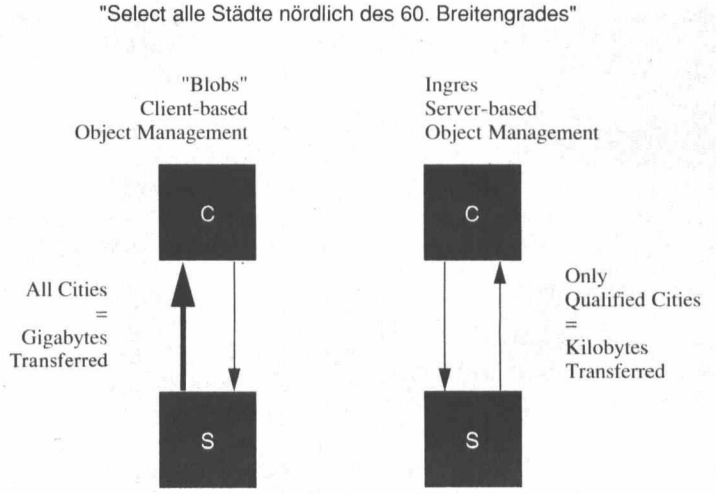
Ein wesentliches Merkmal des Object Managers stellt die Möglichkeit zur Definition eigener Funktionen dar. Die vom Anwender erzeugten Funktionen können sowohl auf die Standarddatentypen (Integer, Floating Point, Strings, Date und Money) als auch auf neue Datenobjekte angewandt werden. Der Anwender kann somit für seine Branche beziehungsweise sein Unternehmen spezifische Datenobjekte und spezifische Funktionen einsetzen.

Das Datenbank-Management-Basismodul

Ingres hat aufbauend auf einer echten Multi-Server-Architektur das Datenbank-Management-System (Backend) für den Einsatz in Hochleistungsanwendungen mit neuen Leistungsmerkmalen ausgestattet. Hierzu gehören ein umfassendes Online-Backup, Two-Phase-Commit mit Multi-Session-Anbindung, logische Schlüssel, Multi-Server-Fast-Commit, nationalisierte Sortieralgorithmen und andere Features. Diese Neuerungen kommen primär Anwendern zugute, die das System in komplexen Applikationen und/oder OLTP-Anwendungen einsetzen.

Online-Backup

Online-Backup erlaubt es dem Anwender, eine Kopie (Backup) von einer Datenbank während des laufenden Transaktionsbetriebs des Datenbanksystems herzustellen. Backup und Produktionsbetrieb laufen gleichzeitig ab. Das darf nicht verwechselt werden mit der Möglichkeit, Datenbankteile vorübergehend gegen Schreibzugriffe zu blockieren, um in den Zeitspannen Backups anzufertigen. Diese einfache Alternative ist teilweise in anderen Datenbanksystemen implementiert. Nutzen aus dem Online-Backup ziehen in erster Linie die Anwender, die den Datenbankbetrieb rund um die Uhr aufrechterhalten müssen. Die Vorteile kommen aber auch Unternehmen zugute, die über extrem umfangreiche Datenbestände verfügen. In diesem Fall würde ein separater Backup-Lauf das Datenbanksystem für eine ungebührlich lange Zeitspanne vom laufenden Betrieb ausschließen.



Two-Phase-Commit und Multiple-Session-Anbindung

Das Leistungsmerkaml des Two-Phase-Commit erlaubt die Durchführung verteilter Transaktionen, die mehrere Datenbanken einbeziehen. Two-Phase-Commit ist für Ingres einer von mehreren Bausteinen bei der Realisierung transparent integrierter Informationssysteme. Andere Komponenten hierfür stellen beispielsweise Ingres/Net und Ingres/Star dar.

Auch die Möglichkeit zur Multiple-Session-Anbindung gehört zum Integrationskonzept. Hierdurch wird es möglich, von einem Applikationsprogramm aus auf mehrere Datenbanken zuzugreifen. Die Umschaltung zwischen den Datenbeständen erfolgt durch eine SQL-Anweisung. Zusammen mit den anderen Integrationskomponenten wird es somit noch einfacher, verteilte Transaktionssysteme mit uneingeschränkter Transaktionsintegrität über mehrere Datenbanken hinweg aufzubauen.

Logische Schlüssel

Mit logischen Schlüsseln läßt sich die eindeutige Identifizierung von Zeilen sicherstellen, die in eine Datenbank eingefügt werden. Die Eindeutigkeit kann sich entweder auf eine einzelne Tabelle oder auf eine komplette Datenbank beziehen. Logische Schlüssel stellen besonders beim Aufbau komplexer objektorientierter Anwendungen ein wichtiges Merkmal dar. ■

Bei herkömmlichen Datenbanksystemen findet eine Objektverwaltung – falls überhaupt – in der Anwendung (Client) statt. Der Datenbank-Manager kann die Objekte nur speichern, aber nicht verarbeiten (sogenannte „Blobs"). Dies ist bei Ingres anders.

Die Programmiersprache C++

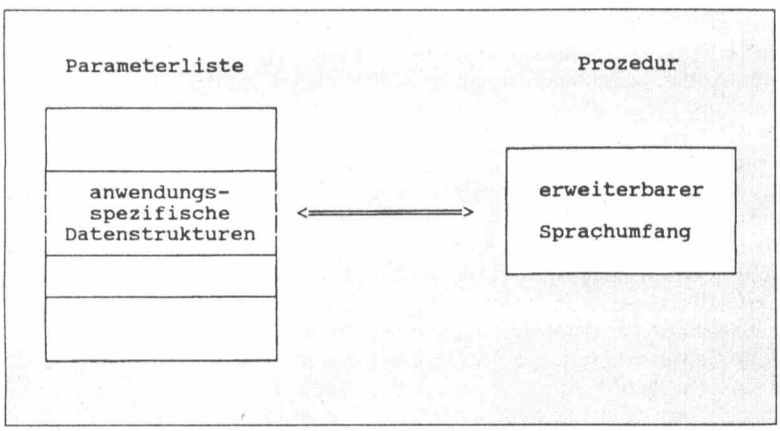
Reiner Kunz

Der Autor
Reiner Kunz studiert derzeit Informatik mit den Nebenfächern Elektrotechnik und Wirtschaft. Sein Arbeitsschwerpunkt ist die Entwicklung von Datenbankapplikationen unter MS-DOS und Unix mit C, C++, dBase und Informix.

Bjarne Stroustrup hätte es als Vater von C++ wohl selbst nicht geglaubt, daß seine Sprachschöpfung eines Tages eine derartige positive Resonanz erfahren wird. Dabei war C++ anfangs gar nicht als eigenständige Sprache, sondern eher als Erweiterung des schon damals bei AT&T entwickelten C gedacht.

Bekanntlich wurde C hauptsächlich aus Gründen der Software-Ergonomie und Entwicklungseffizienz entwickelt. Seine Schöpfer suchten ein passendes Werkzeug, um ein neues Multi-User-Betriebssystem weiterentwickeln zu können. Dieses Unix genannte Betriebssystem war in Assembler geschrieben. Eine Erweiterung der Software machte jedoch die Übertragung in eine höhere Programmiersprache erforderlich, die von Kernighan und Ritchie (kurz K&R) mit C schließlich gefunden wurde. Mit der wachsenden Bedeutung von Unix wuchs auch das Interesse an der Programmiersprache, die dieses Betriebssystem eigentlich erst ermöglichte. Und da C gerade für die Zwecke der Systemprogrammierung viele wichtige Elemente bot, die man sonst nur in Assembler fand, setzten es bald viele Software-Entwickler auf den unterschiedlichsten Computer-Systemen ein. Dank des geringen Sprachumfangs von C konnte der Compiler-Kern relativ leicht an andere Zentraleinheiten angepaßt werden. Die in C geschriebenen Bibliotheken wurden durch erneute Übersetzung auf dem Ziel-Compiler schnell eingesetzt. In C entwikkelte Software war dadurch relativ leicht auf andere Computer-Systeme zu übertragen. Unter anderem liegt es an dieser weitgehenden Portabilität, daß die Sprache C großen Zuspruch bei Software-Entwicklern findet.

Prozedurale Programmierung



Neue Ära der Programmierung

Smalltalk läutete vor über zehn Jahren eine neue Ära in der Geschichte der Programmierung ein. Die bisher verbreitete Programmiermethodik orientierte sich vorwiegend an den Gegebenheiten der prozeduralen Sprachen (Pascal, BASIC, Fortran). Smalltalk bot erstmals die Möglichkeit, Probleme objektorientiert anzugehen. Darüber hinaus ist Smalltalk aber nicht nur eine mächtige Sprache, sondern zugleich eine völlig neue und vor allem eigenständige Entwicklungsumgebung. Ihre Achillesferse ist die benötigte Rechenleistung, sie schränkt ihre Verbreitung zu einem nicht unbeträchtlichen Teil ein.

Erweiterungen

Schon frühzeitig um 1980 entwarf Bjarne Stroustrup bei AT&T Erweiterungen für die Sprache C. Dieses „C mit Klassen" diente zur Behandlung eines Simulationsproblems, wofür ursprünglich SIMULA eingesetzt werden sollte. Man verwarf es jedoch aufgrund seiner geringen Effizienz. „C mit Klassen" fand seine Verwendung in mehreren Simulationsprojekten und übernahm die aus SIMULA bekannten Klassen. Es besaß jedoch noch nicht die von C++ bekannten Fähigkeiten wie das Überladen von Operatoren (overloading), Referenzen und virtuelle Funktionen. Die Verwendung von C als Basissprache hatte dabei mehr pragmatische Gründe: Zum einen kannten die AT&T-Entwickler die Unzulänglichkeiten und Fallstricke dieser Sprache, zum anderen bot C eine überaus effiziente Laufzeitumgebung und erlaubte die Software-Entwicklung unter Unix. Ganz nebenbei existierten bei AT&T umfangreiche C-Source-Codes (etwa Unix) und entsprechend viele C-Programmierer. Somit war zumindestens das Interesse an dieser neuen Spracherweiterung gesichert. Eine eigenständige Sprachentwicklung, unabhängig von C, hätte sicherlich die Lösung einiger Implementierungsprobleme erleichtert, doch wäre C++ dann nur eine weitere unter vielen neuen Programmiersprachen gewesen.

Neue Features

Seit Sommer 1983 ist diese Sprache nun unter der Bezeichnung C++ bekannt und gab der ersten Version einer neuen Sprachgeneration ihren Namen. Die Version 1.1 ergänzte den Sprachumfang um Zeiger auf Member einer Klasse sowie um das Schlüsselwort „protected". Die Version

1.2 bot zusätzlich „unsigned int" und „unsigned long", um überladene Funktionen voneinander zu unterscheiden. Dieser Version hat C++ seine Bekanntheit zu verdanken. Obwohl auch sie noch einige Probleme aufwies, konnte man mit ihr bereits effektiv arbeiten.

Die Entwicklung blieb jedoch nicht stehen. Im Sommer 1989 veröffentlichte AT&T schließlich den Sprachumfang der derzeit aktuellen Version 2.0. Die Erweiterung von C um objektorientierte Fähigkeiten führte zwangsläufig zu Inkompatibilitäten mit dem bestehenden ANSI-C-Standard. Hinzugekommen sind die Schlüsselworte „template" und „catch". Außerdem ist „overload" überflüssig geworden. Seine Verwendung wird durch das sogenannte Type-Safe-Linkage vermieden. Aus Gründen der Abwärtskompatibilität wird dieses Schlüsselwort jedoch weiterhin erkannt.

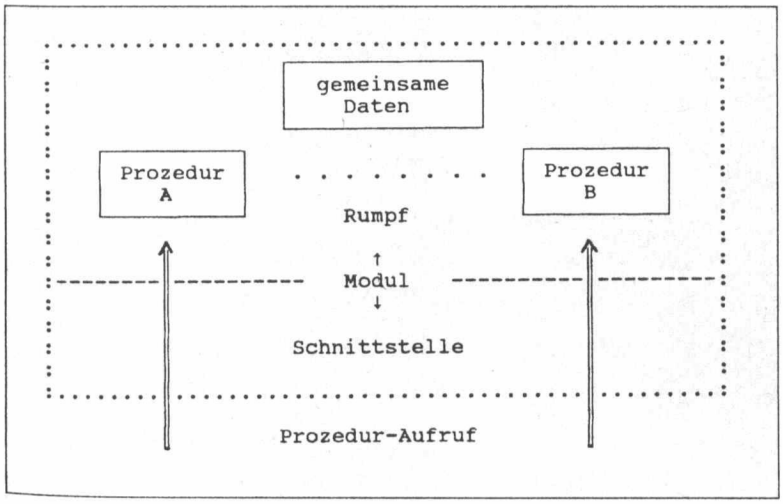
In ANSI-C wandelt „const" eine Variable in eine Konstante, deren Inhalt nicht mehr verändert werden darf. In C++ hingegen ersetzt es den Präprozessor-Befehl „#define" und ermöglicht damit die Typüberprüfung von Konstanten durch den Compiler, während ANSI-C lediglich Texte ersetzt. Mehrfache Definitionen sind in C++ im Gegensatz zu ANSI-C verboten. Ebenfalls verboten ist das Überspringen von Variablen-Initialisierungen per „go to". Würde C++ dies erlauben, könnte die Konstruktion von Objekten übersprungen werden. Ein eventuell im Code vorhandener Destruktor dieses Objekts würde katastrophale Folgen haben.

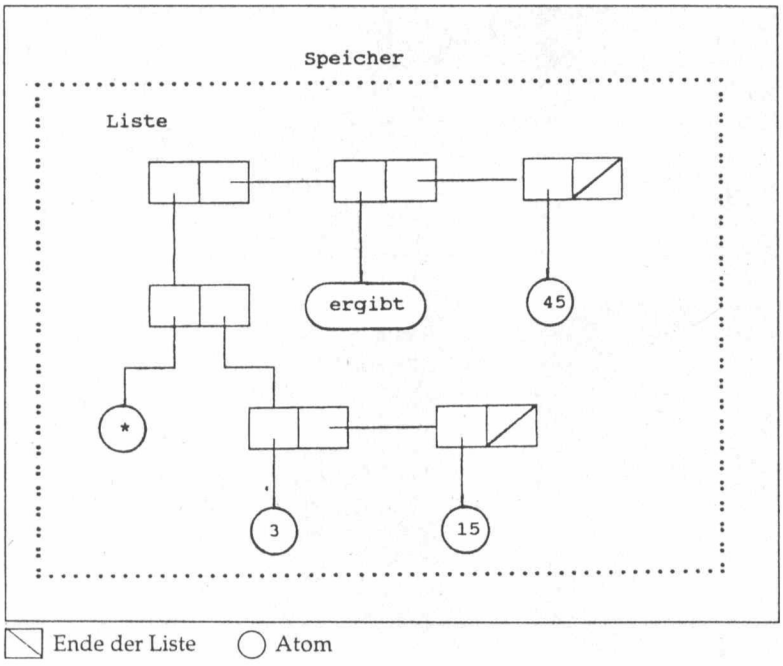
Eine der wichtigsten und grundlegendsten Änderungen der neuen C++-Version ist die mehrfache Vererbung (multiple inheritance). Mit ihr kann eine neue abgeleitete Klasse aus mehr als einer Basisklasse erzeugt werden. Um die daraus resultierenden Mehrdeutigkeiten zu bewältigen, wurde eine neue Terminologie notwendig. So existieren jetzt die Begriffe direkte Basisklasse (direct base class), indirekte Basisklasse (indirect base class) und Unterobjekt (sub-object). Eine direkte Basisklasse steht in der Liste von Basisklassen für eine bestimmte abgeleitete Klasse. Entsprechend erscheint eine indirekte Basisklasse nicht in dieser Liste, wird aber in eine oder mehrere Basisklassen vererbt. Und ein Unterobjekt ist der Speicherbereich, welcher die Basisklasse enthält.

Das Type-Safe-Linkage wurde bereits kurz angesprochen. Frühere C++-Implementierungen kannten diese Einrichtung nicht. Sie hatten speziell beim Einlesen und Übersetzen gewöhnlicher C-Header-Dateien Probleme mit überladenen Funktionsnamen, die nur durch eine Typüberprüfung durch den Linker behoben werden konnten. In der neuen Compiler-Version werden jedoch alle Funktionen mit ausführlichen Informationen über ihre Argumente und Rückgabewerte versehen, so daß bereits zur Compile-Zeit eine Typüberprüfung vorgenommen werden kann. Dieser Vorgang wird als „Mangling" bezeichnet. Da dies jetzt automatisch vom Übersetzer vorgenommen wird, kann man sich die Verwendung von „overload" sparen. Dennoch wird dieses Schlüsselwort von C++ weiterhin erkannt, um die Abwärtskompatibilität zu wahren. Trotzdem kann man dieses Mangling abschalten, um bei Verwendung von gewöhnlichen C-Include-Dateien das korrekte Linken mit den entsprechenden C-Bibliotheken zu ermöglichen.

Ein interessantes Problem ist die Ermittlung der Adressen von überladenen Funktionen. Dazu muß man im Quellcode zusätzlich den Argument-Typ für den Zeiger auf die Funktionsadresse angeben.

Modulare Programmierung

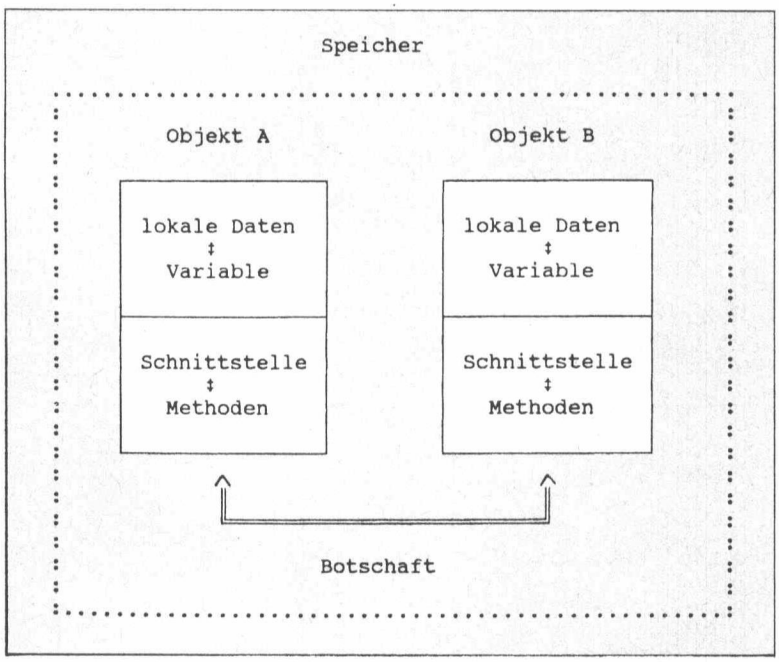




Listenorientierte Programmierung

Speicherreservierung

Die dauerhafte Speicherreservierung für Objekte erfolgt im klassischen C über eine Reihe von Bibliotheksfunktionen (malloc). Verwandte Sprachen wie Pascal und Modula kennen ähnliche Konstrukte, belästigen den Programmierer aber so wenig wie möglich mit technischen Fragen, die sich durch die jeweilige Hardware-Umgebung stellen. Eine sehr komfortable Lösung stellen dazu die beiden Operatoren „new" und „delete" dar. Neben ihnen lassen sich im neuen C++ auch der Kommaoperator sowie der Klassenzeiger „→" überladen.



Objektorientierte Programmierung

Member-Funktionen können jetzt auch „const", „volatile" oder „static" sein. Dabei änderte eine „const"-Member-Funktion keine Member-Daten eines Objekts. Analog dazu kann eine „volatile" Member-Funktion die Member-Daten eines Objektes ändern. Eine weitere Ergänzung des Sprachumfangs betrifft die Behandlung von Fehlerzuständen (exception handling). Während BASIC- und auch dBase-Programmierer mit Hilfe des On-Error-Goto-Konstrukts ihr Programm aus beliebigen Fehlfunktionen zurück in einen bekannten Status führen können, ist das beim klassischen C schon etwas schwieriger. In der Standardbibliothek finden sich dazu die beiden Funktionen setjmp() und longjmp(). Mit setjmp() speichert man die aktuellen Systeminformationen (etwa den Zustand des Kellerspeichers). Abgelegt werden sie in einen Pufferbereich im Speicher, der im allgemeinen mit „jmp_buf" bezeichnet wird. Zu einem beliebigen späteren Zeitpunkt kann nun mit der Bibliotheksfunktion longjmp() dieser Speicherzustand wieder restauriert werden, um bei einem schweren Systemfehler einen definierten Betriebszustand zu erreichen.

In C++ ist ein derartiger Mechanismus jedoch bei weitem schwieriger zu realisieren. Es existieren immerhin nicht nur lokale Variablen auf dem Stack, die man durch Umsetzen des Stackpointers reaktivieren oder löschen kann. Gleichzeitig müßte man den gesamten Stack restaurieren und zusätzlich noch alle Konstruktoren erneut aktivieren, um einen lauffähigen Zustand herzustellen. Da auch in der aktuellen C++-Version dieses Problem noch nicht zufriedenstellend gelöst werden konnte, wurde das Schlüsselwort „catch" für die Fehler- und Ausnahmebehandlung in späteren Compiler-Versionen reserviert.

Weiterentwicklung

Eine wichtige Erweiterung von ANSI-C war die Aufnahme des von Pascal und Modula-2 her bekannten Aufzählungstyps. Auch C++ kennt diesen Typ. Viele Programmierer scheinen eine Abneigung gegen ihn zu haben. Man muß nämlich darauf achten, Aufzählungsnamen nicht mehr als einmal zu benutzen. Gerade in Bibliotheken ist deswegen der Einsatz des Aufzählungstyps zu vermeiden, da man nicht wissen kann, welche Namen der Benutzer verwenden wird. In der neuen C++-Version können die Elemente innerhalb eines Aufzählungstyps lokal zu ihrer Klasse deklariert werden, und man kann den Elementzugriff durch die Schlüsselwörter „private" und „protected" kontrollieren. ■

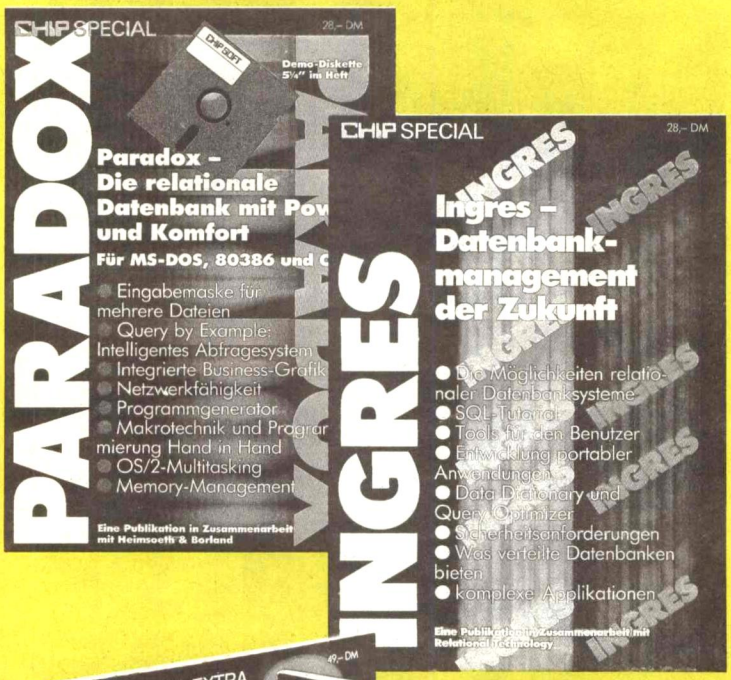
Impressum
CHIP PLUS erscheint monatlich als Beilage zu CHIP.

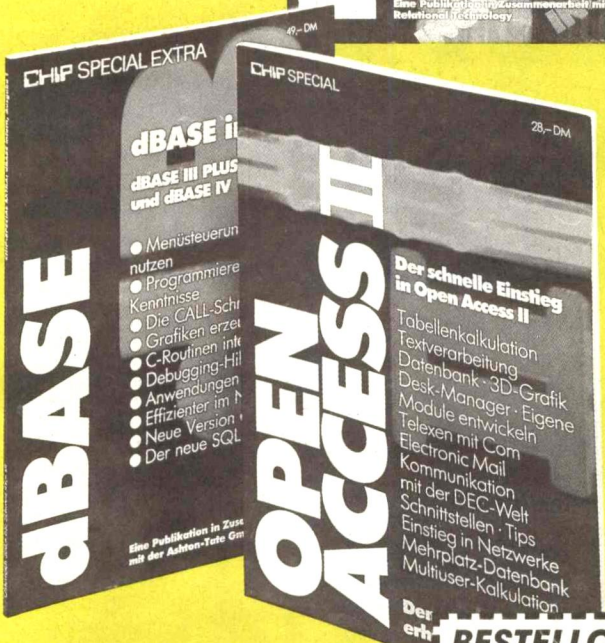
Anschrift der Redaktion: Schillerstr. 23a, 8000 München 2, Tel. 0 89-5 14 93-0.
Verlag: Vogel Verlag und Druck KG, Max-Planck-Str. 7/9, 8700 Würzburg, Tel. 09 31-4 18-0.
Redaktionsdirektor: Dr. Gerald O. Dick
Redaktion: Martin Stübs (verantw.), Ulrich Parthier
Grafische Gestaltung: Hans Kuh, Nadja Schubert
Chef vom Dienst: Reiner Lanninger
Textchef: Dr. Alfred Günter Neumann
Autoren dieser Ausgabe: Günter Jürgensmeier, Alfred Ertl, Silvia Gutschmidt, Rainer Kunz, Prof. Larry Rowe, Ursula Heitfeld
Verlagsdirektor: Dr. Andreas Kaiser
Anzeigen: Friedrich Mangold
Satz, Litho, Druck, Verarbeitung, Versand: Vogel Verlag und Druck KG

Der Verlag haftet nicht für unverlangt eingesandte Manuskripte und Fotos.

CHIP SPECIAL

DATENBANKEN





NEU

Paradox Relationale Datenbank für MS-DOS und OS/2 Mit 5,25'' Demo-Diskette im Heft

Einen tiefen Einblick in die Arbeitsweise dieses modernen relationalen Datenbanksystems unter MS-DOS, 8036 und OS/2 gibt dieses SPECIAL. Die Handhabung und Verknüpfung von Tabellen, der Einsatz und die Funktionsweise der Generatoren wird ebenso wie das Arbeiten mit dem „Query-by-Example" – Abfragesystem ausführlich dargestellt. Anhand zahlreicher Beispiele läßt sich sowohl von Einsteigern als auch von Software-Entwicklern der Umgang mit der PAL-Programmiersprache erlernen. Auch die Anwendung der Netzfunktionen wird erläutert.
Aus dem Inhalt:
* Eingabemaske für mehrere Dateien * Query by Example: Intelligentes Abfragesystem * Integrierte Business-Grafik * Netzwerkfähigkeit * Programmgenerator * Makrotechnik und Programmierung Hand in Hand * OS/2-Multitasking * Memory-Management
Best.-Nr.: 1009, DM/sfr 28,–, öS 230,–

dBase intern dBase III Plus und dBase IV

Viele neue und selbst für Insider unter den dBASE-Anwendern oft unbekannte Möglichkeiten lassen sich mit Hilfe dieser Zusatzfunktionen-Sammlung realisieren. Geboten werden neben nützlichen Tips und Tricks eine Fülle von Utilities, Tools und Erweiterungs-Hilfen. Die dargebotenen Routinen unterstützen die Benutzer des mächtigen Softwarepaketes dBASE III PLUS und IV, sind aber zum überwiegenden Teil auch unter dBASE III lauffähig. Aus dem Inhalt:
* Effektive Menüsteuerung * Einfach programmieren * Die CALL-Schnittstelle – Grafik erzeugen * C-Routinen * Debugging-Hilfen * Anwendungen schützen * Effizienter im Netz * Die neue Version dBASE IV * Der neue SQL-Server * Anbindungen an Minis
Best.-Nr.: 0981, DM/sfr 49,–, öS 380,–

INGRES – Datenbankmanagement der Zukunft

Informationen und ihre Verwaltung in Datenbank-Managementsystemen sind zunehmend von strategischer Bedeutung. Dieses Heft führt Interessierte in die Bedeutung und Funktionsweise von verteilten relationalen Datenbanksystemen ein. In praktischen Beispielen wird demonstriert, wie man portable Anwendungen entwickelt und welche Rolle ein Data Dictionary und ein Query Optimizer spielen. SQL, CASE, Datenbanktuning und Transaktionsverarbeitung sowie Sicherheit und Integrität sind weitere wichtige Themen, die nicht nur Anwendern eine wertvolle Hilfe für ihr Arbeitsgebiet geben.
Best.-Nr.: 1012, DM/sfr 28,–, öS 230,–

Open Access II Der schnelle Einstieg in Open Access II.

Das integrierte Software-Paket Open Access von SPI erfreut sich wachsender Beliebtheit. In seiner Version II mausert es sich zu einer der komfortabelsten Softwarelösungen. Beinahe alle Business-Arbeiten sind besser und schneller zu erledigen. Anhand vieler Beispiele in diesem „schnellen Einstieg" eröffnen sich Ihnen die großen Möglichkeiten von OAII.
Best.-Nr.: 0520, DM/sfr 28,–, öS 230,–

Für Ihre Bestellung verwenden Sie bitte den Coupon oder die Bestellkarten aus diesem Heft. Für ganz Eilige gibt es den telefonischen Tag- und Nacht-Bestellservice: 0931/4182283. Weitere Special-Angebote finden Sie in unserem Katalog, den Sie kostenlos beim Verlag anfordern können.

BESTELLCOUPON:

Bitte ausfüllen, unterschreiben und einsenden an CHIP-SPECIAL-Leser-Service 731, Vogel Verlag und Druck KG, Postfach 6740, D-8700 Würzburg 1

Ja, bitte liefern Sie mir die angekreuzten SPECIAL zu den genannten Preisen plus Versandkosten.

	Stück	Best.-Nr.	Einzelpreis DM/str.	öS
Katalog '89			Gratis	
Paradox		1009	28,–	230,–
dBASE intern		0981	49,–	380,–
Ingres		1012	28,–	230,–
Open Access II		0520	28,–	230,–

Datum, Unterschrift

Vorname, Name

2509

Straße, Nr.

PLZ, Ort

Die Lieferung der SPECIAL erfolgt gegen Rechnung plus Versandkostenanteil.

CHIP Und nichts anderes.

CHIP WISSEN

Die kompetente Buchreihe rund um den PC

Harald Bode
Software-Intensiv-Training – dBase III Plus
376 Seiten, zahlr. Bildschirmfotos, DM 58,–, Hardcover, Großformat, ISBN 3-8023-**0231**-1

Bernd Kretschmer
Multiplan 3.0 auf IBM PC und MS-DOS-Rechnern
mit Multiplan Junior und Windows
382 Seiten, 268 Bilder, DM 48,–, Hardcover, ISBN 3-8023-**0883**-2

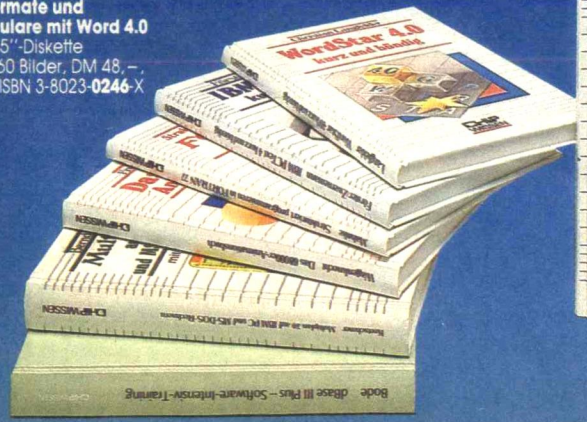
Fred Wagenknecht
Das 68.000er-Animationsbuch
Bewegte Grafik auf Amiga und Atari ST
248 Seiten, 134 Bilder, DM 48,–, Hardcover, ISBN 3-8023-**0211**-7

Hans Mahnke
Strukturiert programmieren in FORTRAN 77
128 Seiten, 20 Bilder, DM 30,–, Hardcover, ISBN 3-8023-**0221**-4

H.-P. Förster/M. Zwernemann
IBM PCText 4 kurz und bündig
196 Seiten, 49 Bilder, DM 38,–, Hardcover, ISBN 3-8023-**0247**-8

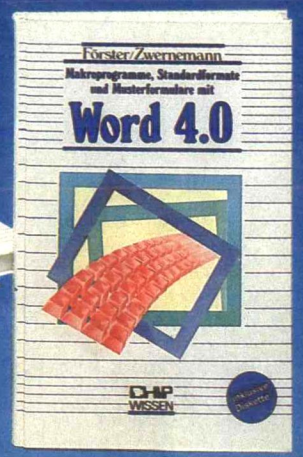
Christian Langfelder
WordStar 4.0 kurz und bündig
mit Mail Merge
184 Seiten, 46 Bilder, DM 35,–, Hardcover, ISBN 3-8023-**0219**-2

H.-P. Förster/M. Zwernemann
Makroprogramme, Standardformate und Musterformulare mit Word 4.0
inklusive 5,25"-Diskette
144 Seiten, 60 Bilder, DM 48,–, Hardcover, ISBN 3-8023-**0246**-X



CHIP
das erfolgreiche Mikrocomputer-Magazin in Europa. CHIP bringt alles, was es an Wissenswertem und Neuem rund um den Bereich des Mikrocomputers gibt: Vergleichstests, Kaufberatung, aktuelle Trends und Tendenzen. Und zeigt auch, was Sie mit einem Computer alles machen können.

Das Verzeichnis **„CHIP Bücher-Sonderhefte-Software 1989"** beschreibt über 150 Titel für Einsteiger, Hobbyprogrammierer und Profis. Auf 36 farbigen Seiten erfahren Sie mehr über die Welt um CHIP. Ihr Buch- und Fachhändler hält Ihr Exemplar für Sie bereit oder einfach anfordern beim Vogel-Verlag, Postfach 67 40, 8700 Würzburg 1



CHIP SPECIAL

Hier eine Auswahl:

UNIX lernen auf dem PC
Für Personal Computer mit MS-DOS-Betriebssystem. Mit diesem Special kann der Umgang mit Unix geübt werden, ohne ein Unix-Betriebssystem oder einen Unix-Rechner zu besitzen. Der größte Teil der Unix-Befehle, soweit sie sich auf dem PC implementieren lassen, wird erklärt und für praktische Übungen zugänglich gemacht. Die vorgestellte Unix-Shell für den PC ist auf Diskette erhältlich.
Best.-Nr. 0988,
DM/sfr. 28,–, öS 230,–.

OS/2 LAN
Das lokale Netzwerk für OS/2 Rechner.
Am Beispiel der 3Com-Netzwerksysteme wird aufgezeigt, was der OS/2-LAN-Manager bietet. Die Vorteile, die durch die Einbettung in MS-OS/2 gegeben sind, die architektonische Verzahnung mit diesem Betriebssystem und die Möglichkeiten dieser Netzwerk-System-Software werden aufgezeigt. Dabei spielt das Konzept der verteilten Applikationen, des Workgroup-Computing und der offenen Plattform eine besondere Rolle.

Aus dem Inhalt:
* Netzwerke im Überblick * OS/2, neue Basis für leistungsfähige Netzwerke * Netzwerk-Planung am Praxisbeispiel * Installieren eines 3+Open-Netzwerkes * Die zahlreichen Verwaltungsfunktionen * Einfache Bedienung mit SAA-Oberfläche und Maus * Neue Möglichkeiten für netzwerkfähige Software * Offene Plattform für die Kommunikation
Best.-Nr. 1000,
DM/sfr. 28,–, öS 230,–.

MS-DOS 3
TDB-3. Die Turbo-Datenbank für kompatible PC. Die preisgünstigste Datenbank, die es auf unserem Markt gibt, ist in dieser Ausgabe mit allen ihren Möglichkeiten komplett enthalten. „Insgesamt ist TDB-3, gemessen am Preis, unschlagbar und stellt in puncto Leistung weit teurere Programme in den Schatten" (Testurteil CHIP 11/88). Darüberhinaus wird aufgezeigt, was sich mit Hilfe von Zusatzfunktionen alles realisieren läßt. Anwenden läßt sich dieses als Diskette beigefügte Programm auf allen IBM-PC und kompatiblen Rechnern.
Best.-Nr. 0969,
DM/sfr. 49,–, öS 380,–.

Programmieren
Die Sonderhefte für Programmentwicklung Reihe CHIP Professional

Programmentwickler und Wissenschaftler zeigen hier anhand nachvollziehbarer Beispiele, wie sie komplexe Probleme angehen. Wenn auch vorwiegend Profis zu Wort kommen, so ist doch alles so aufbereitet, daß alle davon profitieren: Vom Systemprogrammierer bis zum Computerfreak.

Programmieren, Ausg. 1
Best.-Nr. 0965,
DM/sfr. 19,–, öS 150,–.

Programmieren, Ausg. 2
Best.-Nr. 0971,
DM/sfr. 19,–, öS 150,–.

Programmieren, Ausg. 3
Best.-Nr. 0974,
DM/sfr. 19,–, öS 150,–.

Programmieren, Ausg. 4
Aus dem Inhalt:
* HASHING: Das schnellste Suchverfahren – Grundlagen, Praktischer Einsatz, Kollisionsbehandlung * Hashing und Speicheroptimierung * Hashing kombiniert mit anderen Verfahren * Modula-2 Top-Speed-Compiler: eine neue Dimension für PC-Programmierer * Transputer: Möglichkeiten mit parallelen Architekturen * Systemprogrammierer: Große Berufschancen
Best.-Nr. 0987,
DM/sfr. 19,–, öS 150,–.